Computational Mechanics (2021) 67:207–230
https://doi.org/10.1007/s00466-020-01928-9

ORIGINAL PAPER

Hierarchical deep-learning neural networks: finite elements and beyond

Lei Zhang[1] · Lin Cheng[2] · Hengyang Li[2] · Jiaying Gao[2] · Cheng Yu[2] · Reno Domel[3] · Yang Yang[1] · Shaoqiang Tang[1] · Wing Kam Liu[2]

Received: 5 July 2020 / Accepted: 8 September 2020 / Published online: 14 October 2020

Abstract
The hierarchical deep-learning neural network (HiDeNN) is systematically developed through the construction of structured deep neural networks (DNNs) in a hierarchical manner, and a special case of HiDeNN for representing Finite Element Method (or HiDeNN-FEM in short) is established. In HiDeNN-FEM, weights and biases are functions of the nodal positions, hence the training process in HiDeNN-FEM includes the optimization of the nodal coordinates. This is the spirit of r-adaptivity, and it increases both the local and global accuracy of the interpolants. By fixing the number of hidden layers and increasing the number of neurons by training the DNNs, rh-adaptivity can be achieved, which leads to further improvement of the accuracy for the solutions. The generalization of rational functions is achieved by the development of three fundamental building blocks of constructing deep hierarchical neural networks. The three building blocks are linear functions, multiplication, and inversion. With these building blocks, the class of deep learning interpolation functions are demonstrated for interpolation theories such as Lagrange polynomials, NURBS, isogeometric, reproducing kernel particle method, and others. In HiDeNN-FEM, enrichment functions through the multiplication of neurons is equivalent to the enrichment in standard finite element methods, that is, generalized, extended, and partition of unity finite element methods. Numerical examples performed by HiDeNN-FEM exhibit reduced approximation error compared with the standard FEM. Finally, an outlook for the generalized HiDeNN to high-order continuity for multiple dimensions and topology optimizations are illustrated through the hierarchy of the proposed DNNs.

Keywords Neural network interpolation functions · Data-driven · r- and rh-adaptivity · Fundamental building block · Rational functions (i.e. RKPM, NURBS and IGA)

1 Introduction

Machine learning is a process by which computers, when given data, create their own knowledge by identifying patterns in data [1,2]. Deep learning, being a subfield of machine learning, is where computers understand challenging and complex concepts by using and building upon several simpler concepts [2] in a hierarchical fashion. This is achieved by the nonlinear information processing of the deep neural network (DNN). The superior performance of DNN is guaranteed by two core principles: (1) universal approximation theorem [3,4] and (2) backpropagation [5,6]. These two principles allow the approximation of highly nonlinear systems through the optimization of the weights and biases for a predefined loss function.

Inspired by the universal approximation, solving ordinary differential equations (ODEs) and partial differential equa-

Lei Zhang and Lin Cheng: Co-first authors

Reno Domel: Summer 2019 undergraduate research intern at Department of Mechanical Engineering, Northwestern University.

✉ Shaoqiang Tang
maotang@pku.edu.cn

✉ Wing Kam Liu
w-liu@northwestern.edu

[1] HEDPS and LTCS, College of Engineering, Peking University, Beijing 100871, China

[2] Department of Mechanical Engineering, Northwestern University, 2145 Sheridan Rd., Evanston IL 60208-3111, USA

[3] University of Notre Dame, Notre Dame IN 46556, USA

tions (PDEs) through neural networks has been explored in several previous works [7–9], in which shallow neural networks are studied, and fixed mesh is used for the approximation. With the recent breakthrough in deep learning, increasing attention has been paid to the development of DNN-based solutions for the approximation of ODEs and PDEs [10–16]. Most of these recent methodologies solve ODEs or PDEs by randomly sampling points in the domain, defining the loss function as the summation of residuals for governing equations and boundary conditions, and using the DNN for the solution approximation.

In this paper, the Hierarchical Deep-learning Neural Network (HiDeNN) is systematically developed through the construction of structured deep neural networks (DNNs) in a hierarchical manner, and a special case, HiDeNN-FEM is introduced. The HiDeNN-FEM aims to create a DNN that takes in a nodal coordinate as input and produces an associated global shape function with compact support by means of a DNN, whose weights and biases are solely based on the nodal positions. Hence, the training process in HiDeNN-FEM is centered around optimizing nodal positions. This is the spirit of r-adaptivity, which not only increases the local and global accuracy of the interpolants, but also results in a significant reduction of training variables. By fixing the depth of the hidden layers, rh-adaptivity can be achieved by adding neurons based on the required accuracy within the training process. Specifically, three building blocks are designed and systematically combined in order to generate interpolation functions of the finite element method (FEM) [17,18] , splines [19], NURBS [20], isogeometric analysis (IGA) [21–23], reproducing kernel particle method (RKPM) [24–26], and others, making HiDeNN agnostic to the choice of local interpolation functions. Therefore, structured DNNs can produce different interpolation functions that use various discretization strategies in the different regions of a simulated part. Finally, the hierarchical assembly of structured DNNs into one neural network allows for the approximation of a local solution (e.g., nodal displacement) with a coordinate on the simulated part.

In addition, the enrichment in generalized FEM [27], extended FEM [28], and a partition of unity [29] is achieved by multiplying the enrichment functions by the neurons in the region of interests. This leads to the intelligent update of the enrichment regions depending on the object's physics. Furthermore, HiDeNN-FEM utilizes physics-based functions (e.g., principle of minimum potential energy) as the loss function, while the boundary conditions (including both the natural boundary conditions and necessary boundary conditions) are imposed as in the standard FEM manner. Hence, the issues of the imposition of the boundary conditions are avoided, and a more accurate representation of the boundary values is guaranteed. While the standard FEM solves the problem by assembling a stiffness matrix in order to solve the differential equations, the HiDeNN-FEM solves the differential equations by training the DNNs, leading to more flexibility in terms of interpolants and discretization.

The structure of this paper is summarized as follows. In Sect. 2, we present the basics of the HiDeNN-FEM approximation including the representation of a linear shape function and rh-adaptivity. In Sect. 3, the three building blocks for developing DNNs for the rational functions are introduced. The extension of HiDeNN-FEM to Lagrangian polynomials, isogeometric analysis, RKPM, as well as enrichment are also derived in this section. In Sect. 4, the training problem as well as several numerical examples are provided to demonstrate the performance of the proposed methodology. In addition, an outlook of the performance of the HiDeNN when applied to the multiple dimensions with high-order continuity and topology optimization is also demonstrated by constructing the hierarchical DNNs. Finally, the conclusion is made in Sect. 5.

2 Basics of hierarchical deep-learning neural network for finite element method (HiDeNN-FEM)

In this section, the interpolation function represented by the structured deep neural networks (DNNs) is developed. For illustrative purposes, the one-dimensional (1D) linear shape function is used to demonstrate basic ideas of HiDeNN-FEM for constructing interpolation functions by weights, biases and activation functions. Following the construction, r-adaptivity and rh-adaptivity will be introduced in Sects. 2.1 and 2.2 to improve solution accuracy.

2.1 HiDeNN-FEM representation for 1D linear shape function

In standard FEM, the computing domain Ω is discretized by a set of nodal points $x_I \in \Omega$. For an internal node x_I, the global shape function can be formulated as the combination of the shape function from the left side of x_I and that from the right side of x_I, refer to Fig. 1, and written as:

$$N_I(x) = \begin{cases} \dfrac{x - x_{I-1}}{x_I - x_{I-1}}, & x_{I-1} \le x \le x_I, \\ \dfrac{x_{I+1} - x}{x_{I+1} - x_I}, & x_I \le x \le x_{I+1}, \\ 0, & elsewhere, \end{cases} \tag{1}$$

where $N_I(x)$ denotes the shape function at node x_I, while x_{I-1} and x_{I+1} are the two neighbor points of the node x_I from the left side and right side, respectively.

In HiDeNN-FEM, the above global shape function is rewritten in a DNN format, which consists of weights, biases,

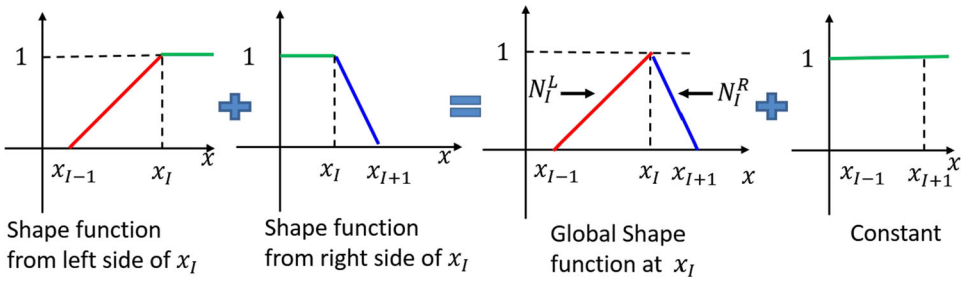
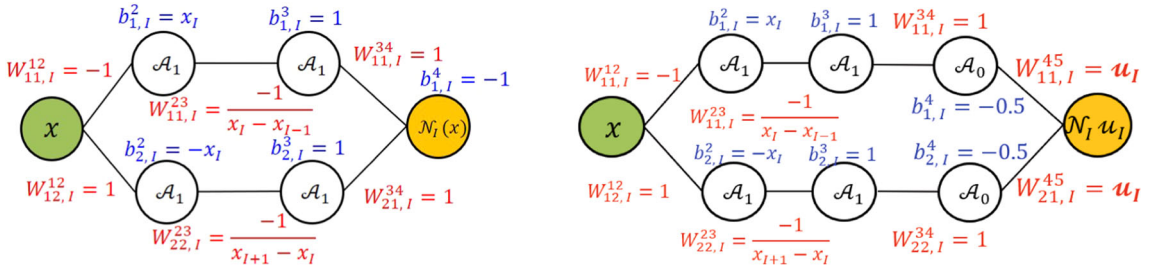


Fig. 1 Assembly of the global shape function at node x_I



(a) DNN-based 1D shape function **(b)** DNN-based 1D interpolation function

Fig. 2 Deep neural network (DNN) representation of the 1D global shape function and interpolation function

Table 1 Notation table of variables used in the DNN representation for 1D shape function in HiDeNN-FEM

$\mathcal{A}_0$	$\mathcal{A}_0(x) = x$
$\mathcal{A}_1$	$\mathcal{A}_1(x) = \max(0,\ x)$
$W^{ij}_{kl,\,I}$	Weights connecting of the k^{th} neuron in layer i to l^{th} neuron in layer j of the DNN for node I
$b^i_{k,\,I}$	Bias of the k^{th} neuron in layer i of the DNN for node I

and activation functions. Figure 2a shows the DNN representation of the linear shape function. It includes two hidden layers, in which the weights and biases are the functions of nodal positions. Based on the DNN representation, the shape function is formulated as

$$\begin{aligned}
&\mathcal{N}_I(x;\, \boldsymbol{W}, \boldsymbol{b},\, \boldsymbol{\mathcal{A}}) \\
&= W^{34}_{11,\,I}\mathcal{A}_1\left(W^{23}_{11,\,I}\mathcal{A}_1\left(W^{12}_{11,\,I}x + b^2_{1,\,I}\right) + b^3_{1,\,I}\right) \\
&\quad + W^{34}_{21,\,I}\mathcal{A}_1\left(W^{23}_{22,\,I}\mathcal{A}_1\left(W^{12}_{12,\,I}x + b^2_{2,\,I}\right) + b^3_{2,\,I}\right) + b^4_{1,\,I} \\
&= \mathcal{A}_1\left(\frac{-1}{x_I - x_{I-1}}\mathcal{A}_1\left(-x + x_I\right) + 1\right) \\
&\quad + \mathcal{A}_1\left(\frac{-1}{x_{I+1} - x_I}\mathcal{A}_1\left(x - x_I\right) + 1\right) - 1,
\end{aligned} \tag{2}$$

where $\boldsymbol{W} = [W^{12}_{11,\,I}, W^{12}_{12,\,I}, W^{23}_{11,\,I}, W^{23}_{22,\,I}, W^{34}_{11,\,I}, W^{34}_{21,\,I}]$, and $\boldsymbol{b} = [b^2_{1,\,I}, b^2_{2,\,I}, b^3_{1,\,I}, b^3_{2,\,I}, b^4_{1,\,I}]$ are the weights and biases of the connected neurons. Detailed definitions of the notations are covered in Table 1. Since the weights and biases in the DNN are functions of nodal positions, the DNN-based shape function can be further expressed as $\mathcal{N}_I(x; \boldsymbol{x}^*_I, \mathcal{A})$, where $\boldsymbol{x}^*_I = [x_{I-1},\ x_I,\ x_{I+1}]$ is the vector that represents the neighbor nodes of node x_I. Once the shape function is obtained, one more hidden layer is added for the interpolation of the displacement associated with the node x_I, which can be written as

$$\begin{aligned}
u^h_I &= \mathcal{N}_I(x;\ \boldsymbol{W}, \boldsymbol{b},\ \boldsymbol{\mathcal{A}})u_I = \mathcal{N}_I(x;\ \boldsymbol{x}^*_I,\ \boldsymbol{\mathcal{A}})u_I;\ \text{no summation on } I \\
&= \mathcal{A}_0\left(\mathcal{A}_1\left(\frac{-1}{x_I - x_{I-1}}\mathcal{A}_1\left(-x + x_I\right) + 1\right) - 0.5\right)u_I \\
&\quad + \mathcal{A}_0\left(\mathcal{A}_1\left(\frac{-1}{x_{I+1} - x_I}\mathcal{A}_1\left(x - x_I\right) + 1\right) - 0.5\right)u_I,
\end{aligned} \tag{3}$$

where u^h_I and u_I are the interpolated displacement and nodal displacement at node x_I, $\boldsymbol{\mathcal{A}} = [\mathcal{A}_0,\ \mathcal{A}_1]$ are the activation functions used for the construction of the DNN approximation. Figure 2b gives the DNN representation of the interpolation of the nodal displacement at node x_I. Unlike the standard FEM, solving the nodal displacements in HiDeNN-FEM is equivalent to the training of the weights at the last hidden layer.

Once the global shape function at the node x_I is represented by the DNN, the DNNs for the entire discretization are obtained by assembling all DNNs together, and then, the problem of interest is solved by training the DNNs. The assembled interpolation for the displacements is thus written as

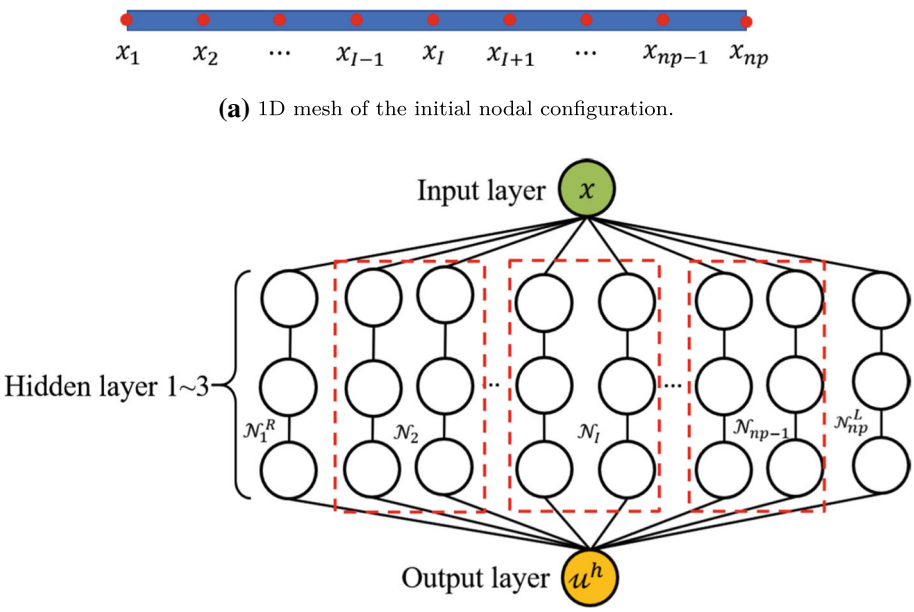


Fig. 3 Deep neural network (DNN) representation of the 1D global shape function and interpolation function

(a) 1D mesh of the initial nodal configuration.

(b) Assembly of the DNNs for the 1D mesh of np nodes

$$u^h(x) = \sum_{I=1}^{np} \mathcal{N}_I(x;\ \boldsymbol{x}_I^*,\ \mathbb{A})u_I, \tag{4}$$

where np represents the total number of nodes within the domain Ω. Figure 3a, b illustrate a 1D mesh and its corresponding DNN interpolation, respectively. As is evident from those two figures, the input of the structured DNN is the nodal coordinates while the output is the interpolated displacements.

Based on the architecture of the DNN representation and solving procedure, we have the following **Observations**:

- Solving the differential equations in HiDeNN-FEM is equivalent to the training of the weights at the last hidden layer.
- In HiDeNN-FEM, weights and biases are functions of the nodal positions, hence the training process in HiDeNN-FEM includes the optimization of the nodal positions. This is the spirit of r-adaptivity and leads to the increase of both the local and global accuracy of the interpolants.
- The r-adaptivity is achieved through a learning process by the structured DNN, and the knowledge is gained through the loss function, i.e., the optimized objective function. The loss function is designed for specific problems. In the following numerical examples, we solve mechanical problems by the variational principle. Thus, here the loss function is the variational formula with respect to the DNN interpolation.

In Fig. 4a, b, we show the change of two weights and two biases, $W_{11,I}^{23}$, $W_{22,I}^{23}$, $b_{1,I}^2$, and $b_{2,I}^2$, is equivalent to the movement of the node x_I, due to update of nodal coordinate from x_I to x_I^*. The associated changes in the DNNs are shown in Fig. 4c, d. The training for the weights and biases in the HiDeNN-FEM is the r-adaptivity in the standard FEM, which is an intrinsic feature of HiDeNN-FEM.

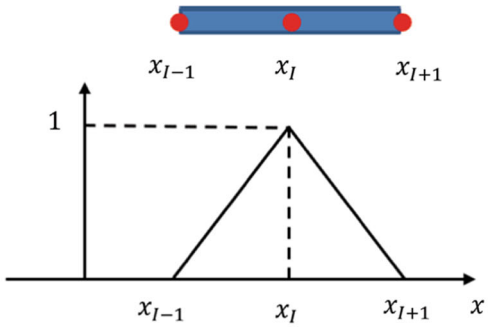
2.2 The rh-adaptivity in HiDeNN-FEM: Adding neurons in neuron networks

In the standard FEM, the h-refinement is achieved by adding new nodes to the existing mesh. This is illustrated in Fig. 5a, c, as the new node $\tilde{x}_K^*$ is added between x_I and x_{I+1}. The region marked by dashed blue box in Fig. 5c is where the extra node is added.

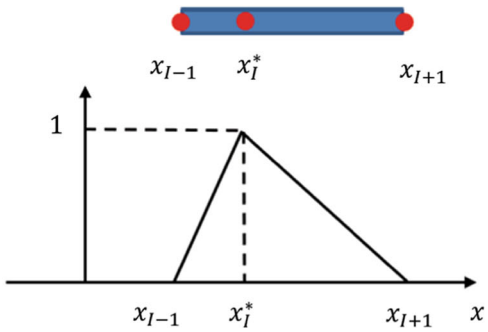
In the framework of HiDeNN-FEM, the h-refinement is the process of adding new neurons from the input layer to the output layer in the global DNN structure of the mesh. For the global DNN, each new node is represented by an extra DNN, as illustrated in Fig. 5d. The newly added DNN for the extra node is marked by the dashed blue box. Since the r-adpativity is an intrinsic feature of HiDeNN-FEM, adding the h-refinement leads to rh-adaptivity for HiDeNN-FEM.

For the rh-adaptivity in HiDeNN-FEM, the following **Observations** are made:

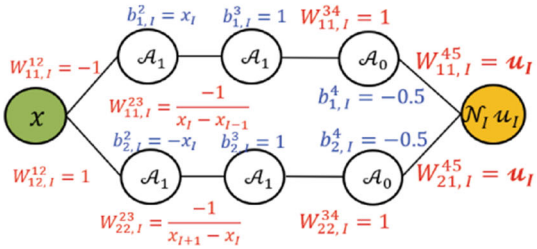
- By fixing the number of hidden layers (e.g., 3 for linear shape functions) and increasing the number of neurons within selected locations, rh-adaptivity can be achieved.
- The h-adaptivity in HiDeNN-FEM is achieved through a learning process involving the training of the neural networks and strategic adding of neurons.



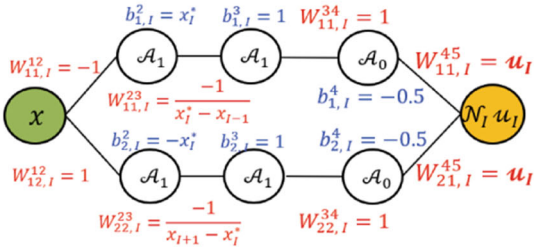
(a) Original shape function for 1D problem



(b) New shape function for 1D problem with r-adaptivity

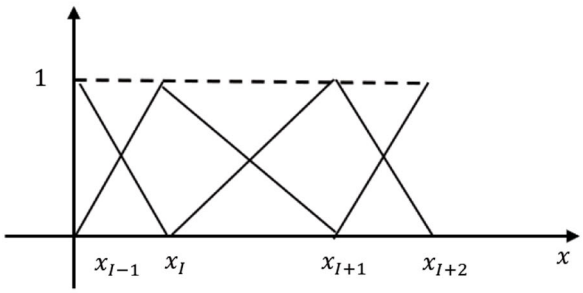


(c) HiDeNN-FEM neuron network for 1D problem

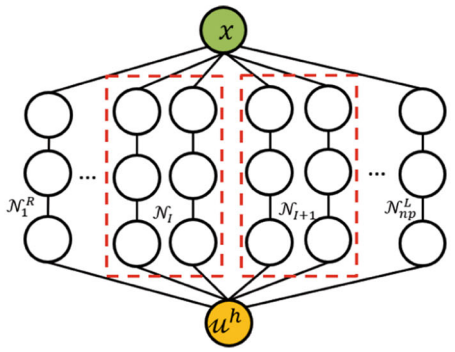


(d) New HiDeNN-FEM neuron network for 1D problem with h-refinement

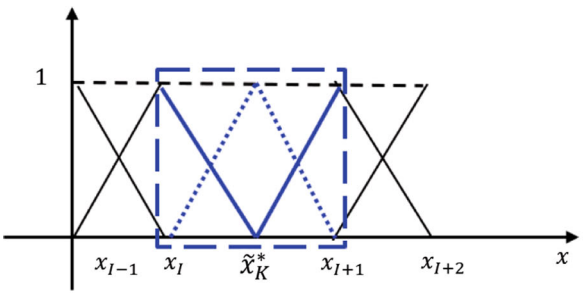
Fig. 4 Illustration of r-adaptivity in HiDeNN-FEM



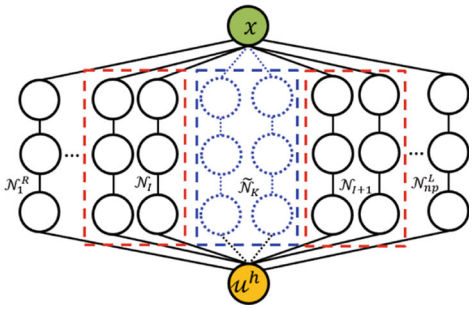
(a) Original shape function for 1D problem



(b) HiDeNN-FEM neuron network for 1D problem



(c) New shape function for 1D problem with h-adaptivity



(d) New HiDeNN-FEM neuron network for 1D problem by the h-adpativity

Fig. 5 Illustration of h-adaptivity in HiDeNN-FEM

Table 2 Notation table of variables used in the elementary blocks and rational functions

$L(x; x_A, x_B, y_A, y_B)$	Regular piecewise linear function
x_A, x_B	Boundaries of linear part of regular piecewise linear function
y_A, y_B	Value at the boundary of linear part of regular piecewise linear function
$\mathscr{L}$	Neural network block corresponding to the regular piecewise linear function
f_1, f_2	Non-negative functions
$\mathscr{F}$	Neural network representation for the non-negative function $f(x)$
$\mathscr{M}$	Neural network multiplication building block
$\mathscr{V}$	Neural network inversion building block
$\mathscr{A}_1$	ReLU activation function defined by $\max(0, x)$
$\mathscr{A}_2$	Quadratic activation function
$\mathscr{A}_3$	Inverse activation function
W_{kl}^{ij}	Weights connecting of the k^{th} neuron in layer i to l^{th} neuron in layer j of the DNN
b_k^i	Bias of the k^{th} neuron in layer i of the DNN

- The accuracy can be improved further by including an adaptivity criterion during the training process.

The h-refinement in the existing FEM is determined by the error estimation, and the mesh is refined in order to obtain convergence results. However, this process does not guarantee the optimal solution. In the HiDeNN-FEM, the criterion for the h-refinement is added into the objective function as a multiplier term and automatically drives the refinement. The criterion will be further elaborated on in Sect. 4. This leads to an optimal strategy compared to the conventional method: the mesh is automatically refined according to the desired scheme, such as fine mesh in stress concentration, heat source, etc. In this paper, we will show how to use the strain magnitude to achieve automatic refinement.

3 DNN representation for rational functions

In this section, the DNN representation for the shape function is further extended to various rational functions, including Lagrange polynomials, B-spline, Reproducing Kernel Particle Method (RKPM), NURBS, Isogeometric analysis (IGA), etc. Specifically, three elementary building blocks are introduced to construct DNNs in a hierarchical manner and simplify the architecture for the description.

3.1 Three elementary building blocks

The notations used in this subsection are summarized in the Table 2.

Linear building block

The first building block is used to formulate the piecewise linear function, which is defined as:

$$L(x; x_A, x_B, y_A, y_B) = \begin{cases} y_A, & x < x_A, \\ \dfrac{y_B - y_A}{x_B - x_A}(x - x_A) + y_A, & x_A \le x \le x_B, \\ y_B, & x > x_B, \end{cases} \tag{5}$$

In DNN representation, the piecewise linear function is rewritten as:

$$\mathscr{L}(x; x_A, x_B, y_A, y_B) = (y_B - y_A)\mathscr{A}_1\left(-\frac{1}{x_B - x_A}\mathscr{A}_1(-x + x_B) + 1\right) + y_A, \tag{6}$$

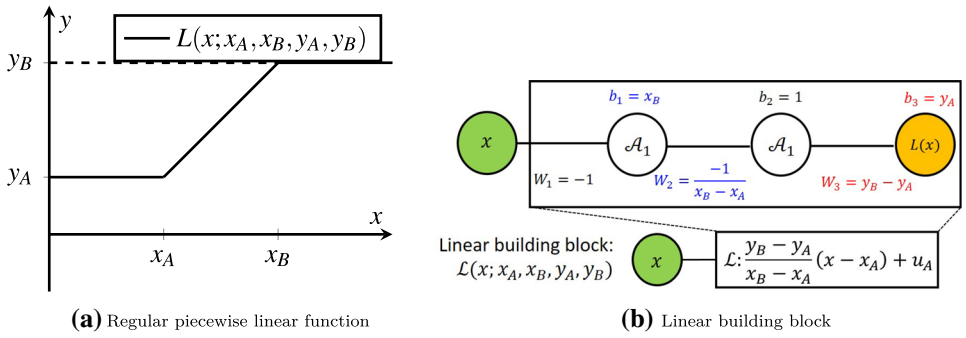
where y_A and y_B are the outputs of linear function at points x_A and x_B, respectively. Figure 6a illustrates the DNN representation of the linear function.

The piecewise linear function is used to capture the local property of the shape function. For instance, when $x_A = x_{I-1}, x_B = x_I, y_A = 0, y_B = 1$, the piecewise linear function becomes the left part of the linear shape function in Eq. (1). The right side of the shape function can be obtained following a similar rule.

Multiplication building block

The second building block is used to achieve multiplication in a DNN. The product of any two non-negative functions f_1 and f_2 is defined as

$$f_1 \cdot f_2 = \frac{1}{2}\left((f_1 + f_2)^2 - f_1^2 - f_2^2\right). \tag{7}$$



(a) Regular piecewise linear function **(b)** Linear building block

Fig. 6 Regular piecewise linear function and corresponding linear building block

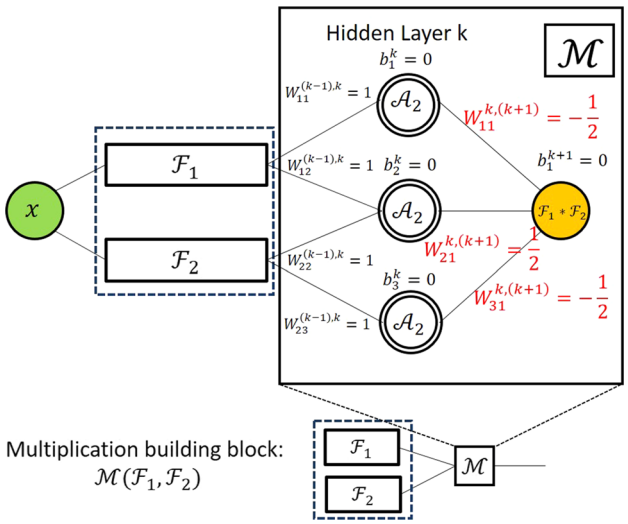


Fig. 7 DNN representation of multiplication building block

In a neural network form, the multiplication can be rewritten by using the activation function as

$$\mathcal{M}(\mathcal{F}_1,\mathcal{F}_2) = \frac{1}{2}\mathcal{A}_2(\mathcal{F}_1+\mathcal{F}_2) - \frac{1}{2}\mathcal{A}_2(\mathcal{F}_1) - \frac{1}{2}\mathcal{A}_2(\mathcal{F}_2), \tag{8}$$

where f_1 and f_2 are two arbitrary non-negative functions, $\mathcal{F}_1, \mathcal{F}_2$ are the corresponding DNN representations, and $\mathcal{A}_2$ denotes the activation function which is defined as

$$\mathcal{A}_2 = \begin{cases} x^2, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{9}$$

Figure 7 illustrates the DNN of the multiplication building block. The inputs are $\mathcal{F}_1$ and $\mathcal{F}_2$, and the output is $\mathcal{F}_1 \cdot \mathcal{F}_2$. The symbol $\boxed{\mathcal{M}}$ is used to represent the multiplication in a DNN and is defined at the bottom of Fig. 7.

Inversion building block

The third building block is to construct the inversion of a given positive function. The inversion block can be used to represent the inversion of a function, as well as performing the division operation (e.g., $a \div b$ is equivalent to $a \times \frac{1}{b}$).

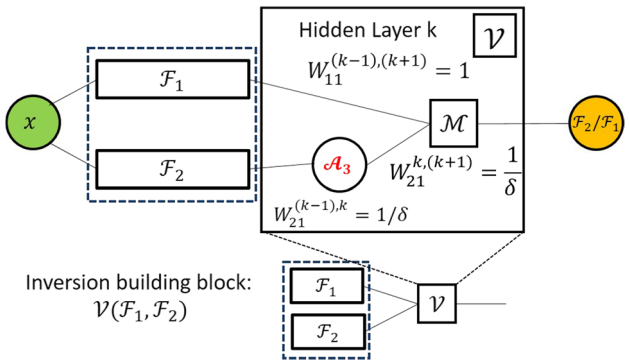


Fig. 8 Neural network representation of inversion building block

First, an activation function $\mathcal{A}_3$ is introduced:

$$\mathcal{A}_3 = \begin{cases} \frac{1}{x}, & x \geq 1, \\ 1, & x < 1. \end{cases} \tag{10}$$

The inversion of a positive function f_2 ($f_2 > \delta > 0$) can thus be expressed as

$$\frac{1}{f_2} = \frac{1}{\delta}\mathcal{A}_3\left(\frac{1}{\delta}f_2\right). \tag{11}$$

In this work, the inversion building block is defined as the multiplication of a non-negative function f_1 with the inversion of the other positive function f_2 ($f_2 > \delta > 0$). It can be formulated in the DNN form as

$$\mathcal{V}(\mathcal{F}_1,\mathcal{F}_2) = \mathcal{M}\left(\mathcal{F}_1, \frac{1}{\delta}\mathcal{A}_3\left(\frac{1}{\delta}\mathcal{F}_2\right)\right), \tag{12}$$

where $\mathcal{F}_1$ and $\mathcal{F}_2$ denote the DNN representation of the functions f_1 and f_2. The DNN representation of the inversion building block is presented in Fig. 8. The boxed symbol $\boxed{\mathcal{V}}$ is used for the representation in the construction.

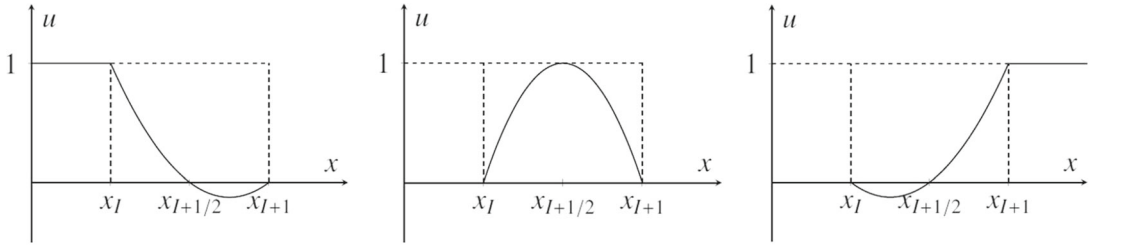


Fig. 9 Quadratic shape functions in the element $[x_I, x_{I+1}]$

3.2 DNN representation for the *p*th order Lagrange shape functions

In this subsection, the three elementary building blocks are utilized to formulate the p^{th} order Lagrange shape functions. By using the linear and multiplication building blocks, the quadratic functions ($p = 2$) are able to be constructed. Figure 9 presents the three second order Lagrange polynomials between the interval $[x_I, x_{I+1}]$, where $\ell_I^p(x)$ denotes the p^{th} order Lagrange functions at the nodal point x_I.

Hence, the three quadratic shape functions at three nodes $x_I, x_{I+\frac{1}{2}}, x_{I+1}$ can be written as:

$$N_I^{p=2}(x) = \ell_I^{p=2}(x),\ N_{I+\frac{1}{2}}^{p=2}(x) = \ell_{I+\frac{1}{2}}^{p=2}(x),$$
$$N_{I+1}^{p=2}(x) = \ell_{I+1}^{p=2}(x), \tag{13}$$

where $N_I^{p=2}(x), N_{I+\frac{1}{2}}^{p=2}(x), N_{I+1}^{p=2}(x)$ represent the shape functions at the three coordinates in Fig. 9, respectively.

Taking $N_{I+1/2}^{p=2}(x)$ as an example, the DNN representation is expressed by using the linear and multiplication building blocks as

$$\begin{aligned}\mathcal{N}_{I+1/2}^{p=2} = &-\frac{1}{(x_{I+1/2}-x_I)(x_{I+1/2}-x_{I+1})}\\ &\mathcal{M}(\mathcal{L}(x; x_I, x_{I+1}, 0, x_{I+1}-x_I),\\ &\mathcal{L}(x; x_I, x_{I+1}, x_{I+1}-x_I, 0)),\end{aligned} \tag{14}$$

where $\mathcal{N}_{I+1/2}^{p=2}$ is the DNN form of $N_{I+1/2}^{p=2}(x)$, $\mathcal{M}$ and $\mathcal{L}$ are the multiplication building block and the linear function building block, respectively. Figure 10 shows the compact DNN structure of $\mathcal{N}_{I+1/2}^{p=2}$. The part in the blue dashed box is the linear block of $(x - x_I)$ while the part in the red dashed box is the linear block of $(x_{I+1} - x)$. Through the multiplication building block performed on the two linear blocks, the DNN representation for $\mathcal{N}_{I+1/2}^{p=2}$ is obtained. Figure 11 gives the interpolation for displacements with detailed DNNs at $x_{I+\frac{1}{2}}$. It can be concluded that the introduction of the building block significantly simplifies the representation of the shape function, and thus the architecture of the DNNs.

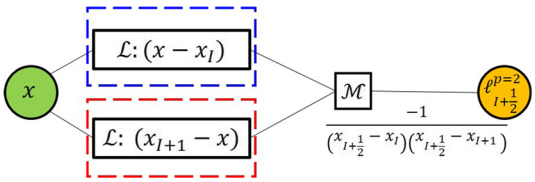


Fig. 10 Compact DNN representation of the quadratic shape function at point $x_{I+\frac{1}{2}}$

Following the same process, the quadratic shape function at points x_I and x_{I+1} is expressed as

$$\begin{aligned}\mathcal{N}_I^{p=2} = &-\frac{1}{(x_I-x_{I+1/2})(x_I-x_{I+1})}\mathcal{M}\left(\mathcal{L}(x; x_I, x_{I+1}, 0, x_{I+1}-x_I),\right.\\ &\left.\mathcal{L}(x; x_I, x_{I+1}, x_{I+1}-x_I, 0)\right)\\ &-\frac{x_I - x_{I+1/2}}{(x_I-x_{I+1/2})(x_I-x_{I+1})}\mathcal{L}(x; x_I, x_{I+1}, x_{I+1}-x_I, 0)\end{aligned} \tag{15}$$

and

$$\begin{aligned}\mathcal{N}_{I+1}^{p=2} = &\frac{1}{(x_{I+1}-x_I)(x_{I+1}-x_{I+1/2})}\\ &\mathcal{M}\left(\mathcal{L}(x; x_I, x_{I+1}, 0, x_{I+1}-x_I),\right.\\ &\left.\mathcal{L}(x; x_I, x_{I+1}, 0, x_{I+1}-x_I)\right)\\ &+\frac{x_I - x_{I+1/2}}{(x_{I+1}-x_I)(x_{I+1}-x_{I+1/2})}\\ &\mathcal{L}(x; x_I, x_{I+1}, 0, x_{I+1}-x_I).\end{aligned} \tag{16}$$

p^{th} order Lagrange polynomials

For p^{th}th order Lagrange shape functions, there are $p + 1$ nodes in the interval $[x_I, x_{I+1}]$, denoted by $x_I, x_{I+1/p}, \cdots,$

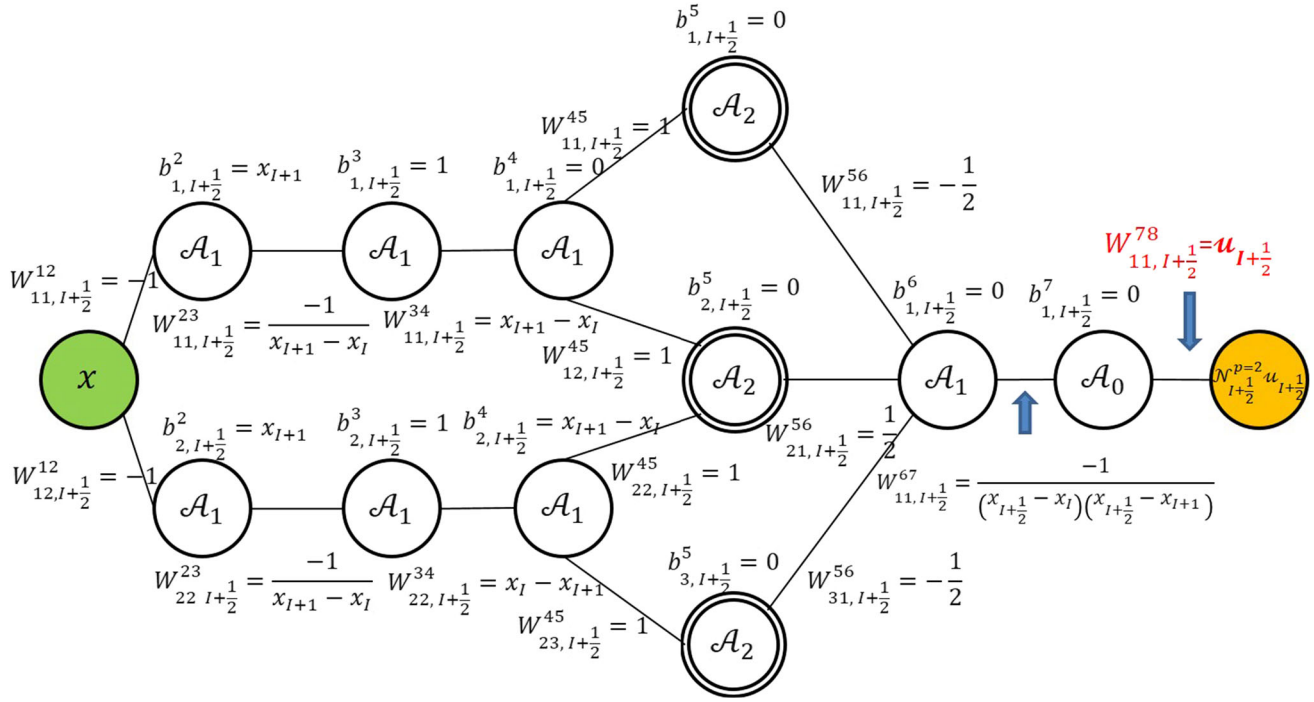


Fig. 11 Detailed DNN interpolation for the quadratic interpolation function at point $x_{I+\frac{1}{2}}$

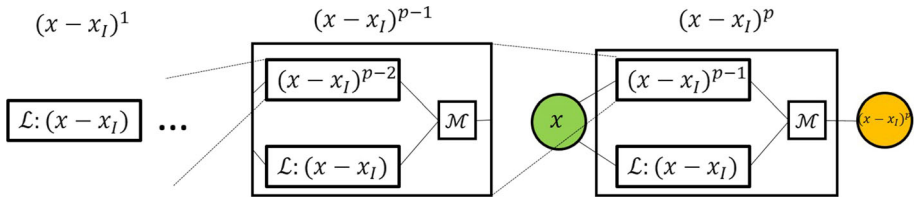


Fig. 12 Recursive way to construct the power function $\mathcal{P}(x,p)$ by the DNN

$x_{I+(p-1)/p}, x_{I+1}$. All shape functions are p^{th} order Lagrange functions, i.e.,

$$l^p_{I+q/p}(x)=\frac{\prod^p_{k=0,k\neq q}(x-x_{I+k/p})}{\prod^p_{k=0,k\neq q}(x_{I+q/p}-x_{I+k/p})}, q=0,1,\cdots,p. \tag{17}$$

These p^{th} order Lagrange functions can be rewritten as

$$l^p_{I+q/p}=a_p(x-x_I)^p+a_{p-1}(x-x_I)^{p-1} +\cdots+a_1(x-x_I)^1+a_0, \tag{18}$$

where $a_k, (k=0,1,\cdots,p)$ are the coefficients, which are the functions of the nodal positions

$$\boldsymbol{x}^*=[x_I, x_{I+1/p}, x_{I+2/p},\cdots,x_{I+(p-1)/p}, x_{I+1}].$$

For the p^{th} order Lagrange function, the DNN representation involves two steps:

(1) Constructing the DNN representation of power functions $\mathcal{P}(x,p)$ according to the recursive rule (refer to Fig. 12):

$$(x-x_I):\mathcal{P}(x,1)=\mathcal{L}(x;x_I,x_{I+1},0,x_{I+1}-x_I); \tag{19}$$
$$(x-x_I)^k:\mathcal{P}(x,k)=\mathcal{M}(\mathcal{P}(x,k-1),\mathcal{P}(x,1)), k\geq 2. \tag{20}$$

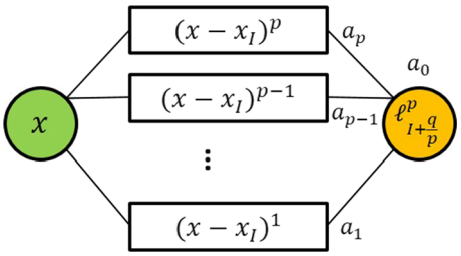


Fig. 13 Summation of power functions for the p^{th} order polynomials

(2) Summing up the power functions $\mathcal{P}(x,k)$ with coefficients $a_k(\boldsymbol{x}^*)$ as shown in Fig. 13.

By introducing the nodal variable $u_{I+q/p}$, the p^{th} order DNN Lagrange interpolation function $\mathcal{N}^p_{I+q/p}u_{I+q/p}$ is illus-

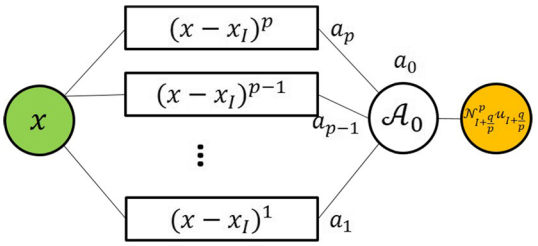


Fig. 14 Nodal interpolation of the p^{th} order polynomials

trated in Fig. 14. Obviously, $\mathcal{N}^p_{I+q/p} u_{I+q/p}$ is determined by the $(p+1)$ nodal positions and the three activation functions $\mathscr{A} = [\mathscr{A}_0, \mathscr{A}_1, \mathscr{A}_2]$.

3.3 DNN representation for B-spline and NURBS basis functions

The B-spline functions as a class of polynomial shape functions [21] are often used in isogeometric analysis. In 1D, p^{th} order B-spline basis functions $B^p_I(x)$ are constructed by the following knot vector $\boldsymbol{k}_x$ in the computed domain $\Omega = [a, b]$,

$$\boldsymbol{k}_x = [x_1 = a, \cdots, x_{n+p+1} = b]^T, \tag{21}$$

where x_I is called as a knot. The initial B-spline basis function $B^0_I(x)$ is a characteristic function

$$B^0_I(x) = \begin{cases} 1, & x_I \le x \le x_{I+1}, \\ 0, & otherwise. \end{cases} \tag{22}$$

We start construction of the DNN from the linear B-spline basis function:

$$\begin{aligned} B^1_I(x) &= \frac{x - x_I}{x_{I+1} - x_I} B^0_I(x) + \frac{x_{I+2} - x}{x_{I+2} - x_{I+1}} B^0_{I+1} \\ &= L(x; x_I, x_{I+1}, 0, 1) + L(x; x_{I+1}, x_{I+2}, 1, 0) - 1. \end{aligned} \tag{23}$$

$\mathscr{B}^1_I$ is used to denote the corresponding DNN representation of $B^1_I(x)$, while $\mathscr{B}^p_I$, $p \in \mathbb{N}$ is used for the p^{th}th order B-spline basis function.

For $p \in \mathbb{N}$, the p^{th} order B-spline basis functions are derived based on the Cox-de Boor formula

$$B^p_I(x) = \frac{x - x_I}{x_{I+p} - x_I} B^{p-1}_I(x) + \frac{x_{I+p+1} - x}{x_{I+p+1} - x_{I+1}} B^{p-1}_{I+1}. \tag{24}$$

By using linear and multiplication building blocks, Eq. (24) is expressed in the DNN form as

$$\begin{aligned} \mathscr{B}^p_I(x) = &\mathscr{M}(\mathscr{L}(x; x_I, x_{I+p}, 0, 1), \mathscr{B}^{p-1}_I(x)) \\ &+ \mathscr{M}(\mathscr{L}(x; x_{I+1}, x_{I+p+1}, 1, 0), \mathscr{B}^{p-1}_{I+1}). \end{aligned} \tag{25}$$

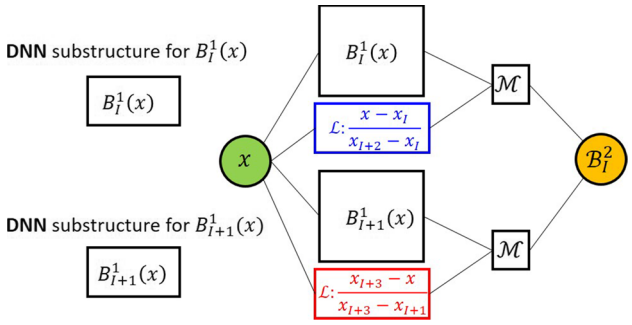


Fig. 15 DNN representation for quadratic B-spline function

Note that the B-spline basis function $B^p_I(x)$ is non-negative and has a compact support $x \in [x_I, x_{I+p+1}]$. So, we can use the multiplication block $\mathscr{M}$, and set linear block $\mathscr{L}$ accordingly.

For instance, the second order B-spline basis function $B^{p=2}_I(x)$ is expressed by:

$$\begin{aligned} B^{p=2}_I(x) = &\frac{x - x_I}{x_{I+2} - x_I} B^{p=1}_I(x) \\ &+ \frac{x_{I+3} - x}{x_{I+3} - x_{I+1}} B^{p=1}_{I+1}(x), \end{aligned} \tag{26}$$

and the correponding DNN representation is:

$$\begin{aligned} \mathscr{B}^{p=2}_I(x) = &\mathscr{M}\left(\mathscr{L}(x; x_I, x_{I+2}, 0, 1), \mathscr{B}^{p=1}_I\right) \\ &+ \mathscr{M}\left(\mathscr{L}(x; x_{I+1}, x_{I+3}, 1, 0), \mathscr{B}^{p=1}_{I+1}\right). \end{aligned} \tag{27}$$

Figure 15 presents the detailed DNN structure for $\mathscr{B}^{p=2}_I(x)$. Blue box is the linear block $\mathscr{L}(x; x_I, x_{I+2}, 0, 1)$, multiplied by the DNN substructure of $\mathscr{B}^{p=1}_I$. And the red box is $\mathscr{L}(x; x_{I+1}, x_{I+3}, 1, 0)$, multiplied by the substructure $\mathscr{B}^{p=1}_{I+1}$. Their combination becomes the second order B-spline basis function $\mathscr{B}^{p=2}_I$.

By definition, a set of the B-spline basis functions forms the partition of unity

$$\sum_{I=1}^{n} B^p_I(x) = 1. \tag{28}$$

The NURBS basis functions are weighted generalizations of B-spline basis functions [21,23]. In particular, a 1D NURBS basis function reads:

$$R^p_I(x) = \frac{B^p_I(x) w_I}{\sum_{J=1}^{n} B^p_J(x) w_J}, \tag{29}$$

where w_I are the weights.

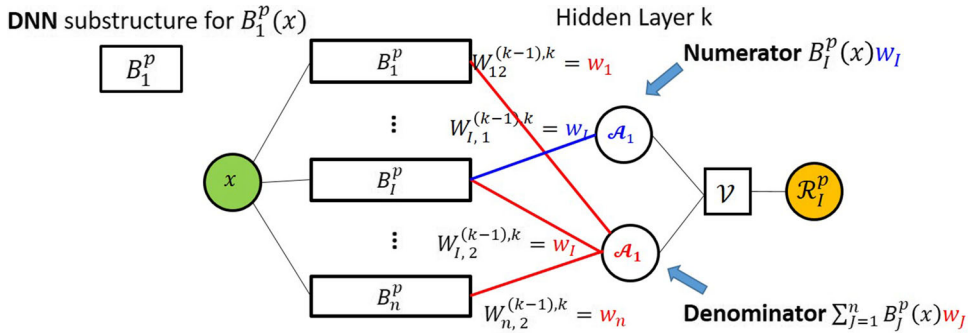


Fig. 16 DNN representation for p^{th} order NURBS function

Any non-negative rational function $R_I^p(x)$ can be represented by using the inversion building block to combine two polynomial functions $B_I^p(x)w_I$ and $\sum_{J=1}^{n} B_J^p(x)w_J$, i.e.,

$$\mathcal{R}_I^p(x) = \mathcal{V}(\mathcal{B}_I^p(x)w_I, \sum_{J=1}^{n} \mathcal{B}_J^p(x)w_J) \tag{30}$$

The DNN representation of the NURBS functions is shown in Fig. 16.

3.4 DNN representation for RKPM basis function

Considering the meshfree shape functions formulated in a domain Ω with np nodes $x_I, I = 1, 2, \cdots, np$. A kernel function $\phi(x^* - x)$ is centered at point x^*, with a compact influence or support domain measured by a support size s.

The meshfree basis function $\Phi_I(x)$ is formulated based on the reproducing kernel approximation theory [24] given by

$$\Phi_I(x) = \boldsymbol{p}^T(x_I)\boldsymbol{M}^{-1}\boldsymbol{p}(x)\phi(x_I - x), \tag{31}$$

where $\boldsymbol{p}(x)$ is the power order basis vector and is defined as

$$\boldsymbol{p}(x) = [1, x, x^2, \cdots, x^p]^T, \tag{32}$$

and the moment matrix is defined as

$$\boldsymbol{M}(x) = \sum_{I=1}^{np} \boldsymbol{p}(x_I)\boldsymbol{p}^T(x_I)\phi(x_I - x). \tag{33}$$

The reproducing basis functions $\Phi_I(x), I = 1, 2, \cdots, np$ defined in Eq. (31) satisfy the reproducing conditions:

$$\sum_{I=1}^{np} \Phi_I(x)\boldsymbol{p}(x_I) = \boldsymbol{p}(x). \tag{34}$$

Now we include the zeroth order case corresponding to the partition of unity:

$$\sum_{I=1}^{np} \Phi_I(x) = 1. \tag{35}$$

The analytical expression of Eq. (31) [30] is

$$\Phi_I(x) = \frac{\sum_{1\le l_1<l_2<\cdots<l_{p-1}\le np} H_{l_1 l_2\cdots l_{p-1}I} H_{l_1 l_2\cdots l_{p-1}p}\phi_{l_1}\phi_{l_2}\cdots\phi_{l_{p-1}}\phi_I}{\sum_{1\le l_1<l_2<\cdots<l_p\le np}(H_{l_1 l_2\cdots l_p})^2\phi_{l_1}\phi_{l_2}\cdots\phi_{l_p}}. \tag{36}$$

Here, we denote $\phi_I = \phi_I(x_I - x)$,

$$H_{l_1 l_2\cdots l_{p-1}I} = \det\left(\left[\boldsymbol{p}(x_{l_1}), \boldsymbol{p}(x_{l_2}), \cdots, \boldsymbol{p}(x_{l_{p-1}}), \boldsymbol{p}(x_I)\right]\right), \tag{37}$$

$$H_{l_1 l_2\cdots l_{p-1}p} = \det\left(\left[\boldsymbol{p}(x_{l_1}), \boldsymbol{p}(x_{l_2}), \cdots, \boldsymbol{p}(x_{l_{p-1}}), \boldsymbol{p}(x)\right]\right), \tag{38}$$

$$H_{l_1 l_2\cdots l_{p-1}l_p} = \det\left(\left[\boldsymbol{p}(x_{l_1}), \boldsymbol{p}(x_{l_2}), \cdots, \boldsymbol{p}(x_{l_{p-1}}), \boldsymbol{p}(x_{l_p})\right]\right). \tag{39}$$

Notice that $H_{l_1 l_2\cdots l_{p-1}p}$ and kernel functions are functions of input variable x. $H_{l_1 l_2\cdots l_{p-1}I}$ and $H_{l_1 l_2\cdots l_p}$ are the coefficients, which are functions of nodal positions $\boldsymbol{x}^* = [x_{l_1}, x_{l_2}, \cdots, x_{l_p}, x_I]$.

In this work, we take non-negative bounded polynomial functions as the kernel function represented by the DNNs. The vector $\boldsymbol{p}(x)$ and moment matrix $\boldsymbol{M}(x)$ are also polynomials. Combining the inversion operator, the RKPM basis function can be represented accordingly.

For example, for an equidistant partition of domain Ω and a support size $s = 4h$ with the grid size h, Fig. 17 gives a kernel function

$$\phi(x) = \left|\frac{x}{h}\right|^3 - 3\left|\frac{x}{h}\right|^2 + 4 \tag{40}$$

in the support domain $[-2h, 2h]$. We only consider the linear basis vector

$$\boldsymbol{p}(x) = [1 \quad x]^T. \tag{41}$$

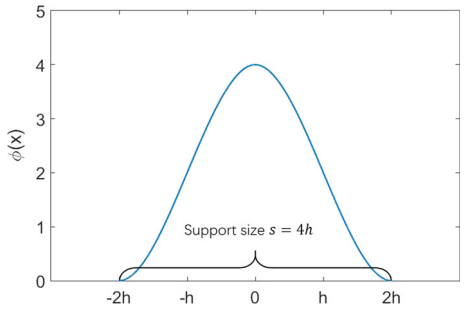


Fig. 17 An example of kernel function: $\phi(x) = \left|\frac{x}{h}\right|^3 - 3\left|\frac{x}{h}\right|^2 + 4$

The right part of basis functions associated with node x_I are

$$\Phi_I(x) = \boldsymbol{p}^T(x_I)\boldsymbol{M}^{-1}(x)\boldsymbol{p}(x)\phi(x_I - x) = \begin{cases} \dfrac{\sum_{k=I-1}^{I+2}(x_I - x_k)(x - x_k)\phi_k\phi_I}{\sum_{k=I-1}^{I+1}\sum_{l=k+1}^{I+2}(x_k - x_l)^2\phi_k\phi_l}, & x \in [x_I, x_{I+1}], \\ \dfrac{\sum_{k=I}^{I+3}(x_I - x_k)(x - x_k)\phi_k\phi_I}{\sum_{k=I}^{I+2}\sum_{l=k+1}^{I+3}(x_k - x_l)^2\phi_k\phi_l}, & x \in (x_{I+1}, x_{I+2}], \\ 0, & x \in (x_{I+2}, +\infty). \end{cases} \tag{42}$$

The left part is obtained in the same way.

The reproducing functions are rational, and the corresponding DNN representations are given by

$$\begin{aligned} &\mathscr{V}\Big((x_I - x_{I-1})\mathscr{M}\Big(\mathscr{L}(x; x_I, x_{I+1}, x_I - x_{I-1}, x_{I+1} - x_{I-1}), \phi_{I-1}\phi_I\Big) \\ &+(x_{I+1} - x_I)\mathscr{M}\Big(\mathscr{L}(x; x_I, x_{I+1}, x_{I+1} - x_I, 0), \phi_{I+1}\phi_I\Big) \\ &+(x_{I+2} - x_I)\mathscr{M}\Big(\mathscr{L}(x; x_I, x_{I+1}, x_{I+2} - x_I, x_{I+2} - x_{I+1}), \phi_{I+2}\phi_I\Big), \\ &\sum_{k=I-1}^{I+1}\sum_{l=k+1}^{I+2}(x_k - x_l)^2\phi_k\phi_l\Big), \\ &x \in [x_I, x_{I+1}] \end{aligned} \tag{43}$$

and

$$\begin{aligned} &\mathscr{V}\Big((x_{I+2} - x_I)\mathscr{M}\Big(\mathscr{L}(x; x_{I+1}, x_{I+2}, x_{I+2} - x_{I+1}, 0), \phi_{I+2}\phi_I\Big) \\ &+(x_{I+3} - x_I)\mathscr{M}\Big(\mathscr{L}(x; x_{I+1}, x_{I+2}, x_{I+3} - x_{I+1}, x_{I+3} - x_{I+2}), \phi_{I+3}\phi_I\Big), \\ &\sum_{k=I}^{I+2}\sum_{l=k+1}^{I+3}(x_k - x_l)^2\phi_k\phi_l\Big) \\ &-\mathscr{V}\Big((x_{I+1} - x_I)\mathscr{M}\Big(\mathscr{L}(x; x_{I+1}, x_{I+2}, 0, x_{I+2} - x_{I+1}), \phi_{I+1}\phi_I\Big), \\ &\sum_{k=I}^{I+2}\sum_{l=k+1}^{I+3}(x_k - x_l)^2\phi_k\phi_l\Big), \\ &x \in (x_{I+1}, x_{I+2}]. \end{aligned} \tag{44}$$

Figure 18 shows the curve of $\Phi_I(x)$ and the corresponding DNN representation of the part in sub-interval $[x_I, x_{I+1}]$.

It is shown that if taking the support size of kernel shape functions as $s = h$ and the basis vector as $\boldsymbol{p}(x) = [1, x]^T$. It is actually the same as the linear global shape function $N_I^{p=1}(x)$ in Sect. 2.1.

3.5 DNN representation for three-dimensional RKPM

The three-dimensional RKPM can also be constructed to solve problems, i.e. in $\Omega = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$ with np unstructured nodes $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_{np}$. Taking the basis vector as

$$\boldsymbol{p}(\boldsymbol{x}) = [1, x, y, z]^T, \tag{45}$$

the kernel function associated with the node $\boldsymbol{x}_I$ is

$$\phi_I = \phi_I(x - x_I, y - y_I, z - z_I). \tag{46}$$

Following Liu et al. [24], the meshfree shape function associated with the node $\boldsymbol{x}_I$ takes the following form

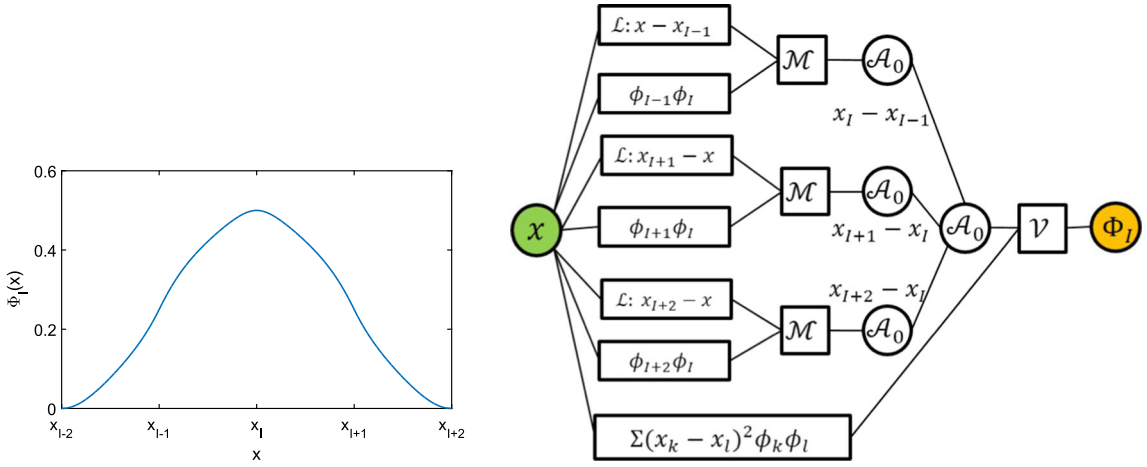
$$\Phi_I(x) = \boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{M}^{-1}(\boldsymbol{x})\boldsymbol{p}(\boldsymbol{x})\phi_I,\ I = 1, 2, \cdots, np, \tag{47}$$

where the moment matrix $\boldsymbol{M}(\boldsymbol{x})$ is

$$\boldsymbol{M}(\boldsymbol{x}) = \sum_{J=1}^{np}\boldsymbol{p}(\boldsymbol{x}_J)\boldsymbol{p}^T(\boldsymbol{x}_J)\phi_J. \tag{48}$$

The corresponding analytical expression [30] is

$$\Phi_I(\boldsymbol{x}) = \frac{\sum_{1\le l_1<l_2<l_3\le np} H_{l_1l_2l_3I}H_{l_1l_2l_3p}\phi_{l_1}\phi_{l_2}\phi_{l_3}\phi_I}{\det(\boldsymbol{M})} = \frac{\sum_{1\le l_1<l_2<l_3\le np} H_{l_1l_2l_3I}H_{l_1l_2l_3p}\phi_{l_1}\phi_{l_2}\phi_{l_3}\phi_I}{\sum_{1\le l_1<l_2<l_3<l_4\le np}(H_{l_1l_2l_3l_4})^2\phi_{l_1}\phi_{l_2}\phi_{l_3}\phi_{l_4}}, \tag{49}$$



(a) RKPM basis function $\Phi_I(x)$ **(b)** Part of neural network representation

Fig. 18 RKPM basis function $\Phi_I(x)$ and the corresponding neural network representation in the element $[x_I, x_{I+1}]$

where we denote

$$H_{l_1l_2l_3I} = \det\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_{l_1} & x_{l_2} & x_{l_3} & x_I \\ y_{l_1} & y_{l_2} & y_{l_3} & y_I \\ z_{l_1} & z_{l_2} & z_{l_3} & z_I \end{bmatrix},$$

$$H_{l_1l_2l_3p} = \det\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_{l_1} & x_{l_2} & x_{l_3} & x \\ y_{l_1} & y_{l_2} & y_{l_3} & y \\ z_{l_1} & z_{l_2} & z_{l_3} & z \end{bmatrix},$$

$$H_{l_1l_2l_3l_4} = \det\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_{l_1} & x_{l_2} & x_{l_3} & x_{l_4} \\ y_{l_1} & y_{l_2} & y_{l_3} & y_{l_4} \\ z_{l_1} & z_{l_2} & z_{l_3} & z_{l_4} \end{bmatrix}. \tag{50}$$

Note that $H_{l_1l_2l_3p}$ and kernel functions $\phi_I (I = 1, 2, \cdots, np)$ are the functions of variables x, y, z. $H_{l_1l_2l_3I}$ and $H_{l_1l_2l_3l_4}$ are the coefficients determined by nodal positions.

The development of deep learning involves two steps. The first step is to rewrite $H_{l_1l_2l_3p}$ as a linear combination of non-negative terms in order to apply the multiplication building block,

$$\begin{aligned} H_{l_1l_2l_3p} &= a_0 + a_1(x - x_{min}) \\ &\quad + a_2(y - y_{min}) + a_3(z - z_{min}), \boldsymbol{x} \in \Omega. \end{aligned} \tag{51}$$

Here, a_0, a_1, a_2, a_3 are all the functions of nodal coordinates. The second step is to construct the neural network using three building blocks.

Figure 19 illustrates $\Phi_I(\boldsymbol{x})$ with only one term in the numerator

$$\Phi_I(\boldsymbol{x}) = \frac{H_{l_1l_2l_3I} H_{l_1l_2l_3p}\phi_{l_1}\phi_{l_2}\phi_{l_3}\phi_I}{\det(\boldsymbol{M})}. \tag{52}$$

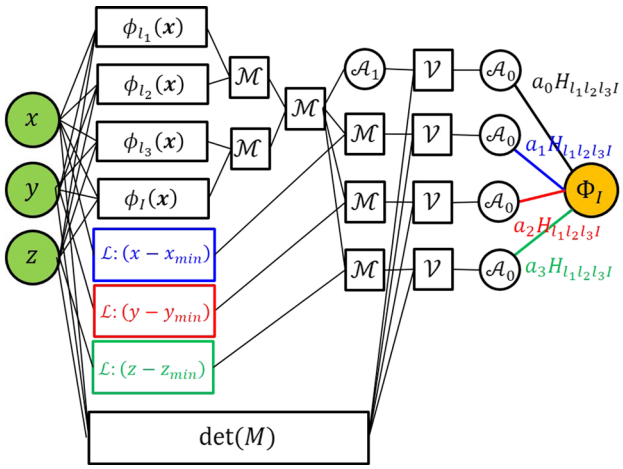


Fig. 19 Illustration for $\Phi_I(\boldsymbol{x})$ with only one term $H_{l_1l_2l_3I} H_{l_1l_2l_3p}\phi_{l_1}\phi_{l_2}\phi_{l_3}\phi_I$ in the numerator

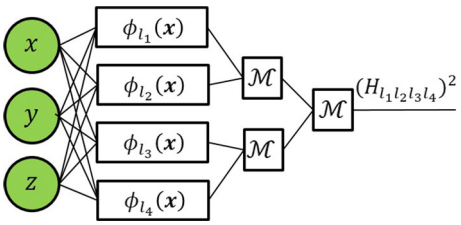
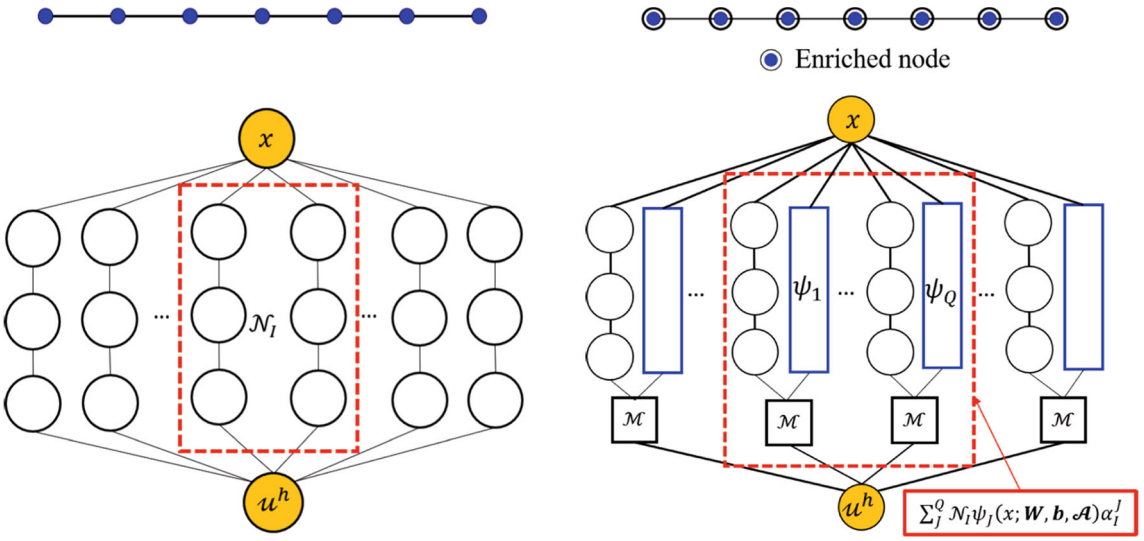


Fig. 20 Illustration for one term $(H_{l_1l_2l_3l_4})^2\phi_{l_1}\phi_{l_2}\phi_{l_3}\phi_{l_4}$ in the denominator of $\Phi_I(\boldsymbol{x})$

The box with "det($\boldsymbol{M}$)" in Fig. 19 represents the substructure of the denominator. Figure 20 illustrates one term in the denominator

$$(H_{l_1l_2l_3l_4})^2\phi_{l_1}\phi_{l_2}\phi_{l_3}\phi_{l_4}. \tag{53}$$



(a) Original HiDeNN-FEM neuron network for 1D problem

(b) New HiDeNN-FEM neuron network for 1D problem with enrichments

Fig. 21 Illustration of enrichment in HiDeNN-FEM

3.6 Partition of unity for the enrichment by HiDeNN-FEM

The partition of unity method (PUM) has been widely accepted in FEM or meshfree method for solving the PDEs with non-smooth solutions (i.e. discontinuities, singularities, high gradient, etc.). The primary goal of the PUM is to extend the standard approximation space to the problems of interest by using previously established enrichment functions. In HiDeNN-FEM, the PUM can be formulated as

$$u^h(\boldsymbol{x}) = \sum_I^{np} \sum_J^Q \mathcal{N}_I \psi_J(x; \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{A}) \boldsymbol{a}_J^I, \tag{54}$$

where $u^h(\boldsymbol{x})$ is the globally approximated solution by the proposed HiDeNN-FEM, the subscripts I is the nodal index, $\mathcal{N}_I$ denotes the DNN shape function at node I, $\psi_J(x; \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{A})$ denotes the enrichment function, which incorporates the prior knowledge into the approximation space. In the HiDeNN-FEM, the enrichment function is constructed by the DNNs, in which $\boldsymbol{W}$, $\boldsymbol{b}$ and $\boldsymbol{A}$ represent the matrix of weights, matrix of biases and activation functions used for the construction of enrichment functions. $\boldsymbol{a}_J^I$ represents the additional unknowns added by the enrichment on node I. J and Q are the indices of enrichment functions and the total number of enrichments applied at the node I. In Eq. (54), the functions $\mathcal{N}_I(\boldsymbol{x})$ build a partition of unity

$$\sum_I^{np} \mathcal{N}_I(\boldsymbol{x}) = 1. \tag{55}$$

Figure 21 presents DNNs of HiDeNN-FEM before and after enrichments for a 1D problem. It is evident that, in HiDeNN-FEM, the enrichment is achieved by using the multiplication building block to multiply the DNNs of enrichment functions with the nodal DNN. Specifically, the bottom right corner of Fig. 21 illustrates the enrichment for nodal I with Q number of enrichment functions $\psi_J(x; \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{A})$.
Based on the construction of the enrichment in HiDeNN-FEM, we have the following **Observations**:

- In HiDeNN-FEM, building an enrichment function through the multiplication of neurons is equivalent to the enrichment in standard finite element method, such as GFEM, XFEM, PUFEM, etc.
- The multilevel of enrichments can be achieved by expanding the substructure of the neural network for each node.
- The enrichment could be achieved by a learning process and combined with r and h-adaptivity to obtain better performance.

3.7 DNN representation for two-dimensional and three-dimensional element shape functions of HiDeNN-FEM

We construct 2D and 3D interpolations based on elements for HiDeNN-FEM. Note the 4-node element in the two-dimensional case as shown in Fig. 22. $\boldsymbol{x}_k^e = (x_k^e, y_k^e), k = 1, 2, 3, 4$ denotes the physical coordinates of 4 nodes of the element. Considering the DNN structure with (x, y) as

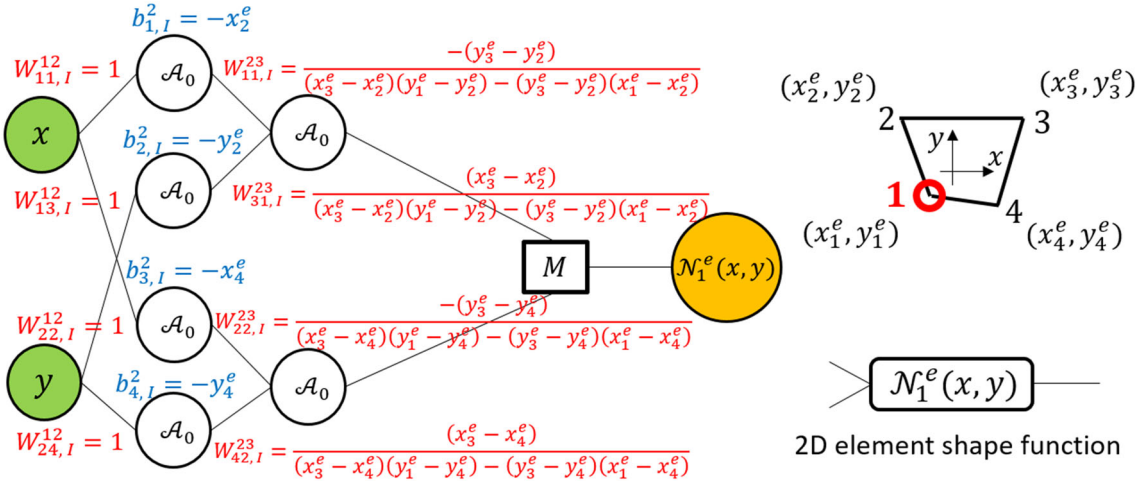


Fig. 22 DNN representation for 2D element shape function $N^e_1(x, y)$. The input is physical coordinates (x, y)

inputs, we construct the DNN representation of the following bi-linear element shape functions in physical coordinates directly,

$$
\begin{aligned}
N^e_1(x,y) &= \frac{(x^e_3 - x^e_2)(y - y^e_2) - (y^e_3 - y^e_2)(x - x^e_2)}{(x^e_3 - x^e_2)(y^e_1 - y^e_2) - (y^e_3 - y^e_2)(x^e_1 - x^e_2)} \\
&\quad \frac{(x^e_3 - x^e_4)(y - y^e_4) - (y^e_3 - y^e_4)(x - x^e_4)}{(x^e_3 - x^e_4)(y^e_1 - y^e_4) - (y^e_3 - y^e_4)(x^e_1 - x^e_4)}, \\
N^e_2(x,y) &= \frac{(x^e_4 - x_3)(y - y^e_3) - (y^e_4 - y^e_3)(x - x^e_3)}{(x^e_4 - x^e_3)(y^e_2 - y^e_3) - (y^e_4 - y^e_3)(x^e_2 - x^e_3)} \\
&\quad \frac{(x^e_4 - x^e_1)(y - y^e_1) - (y^e_4 - y^e_1)(x - x^e_1)}{(x^e_4 - x^e_1)(y^e_2 - y^e_1) - (y^e_4 - y^e_1)(x^e_2 - x^e_1)}, \\
N^e_3(x,y) &= \frac{(x^e_1 - x^e_4)(y - y^e_4) - (y^e_1 - y^e_4)(x - x^e_4)}{(x^e_1 - x^e_4)(y^e_3 - y^e_4) - (y^e_1 - y^e_4)(x^e_3 - x^e_4)} \\
&\quad \frac{(x^e_1 - x^e_2)(y - y^e_2) - (y^e_1 - y^e_2)(x - x^e_2)}{(x^e_1 - x^e_2)(y^e_3 - y^e_2) - (y^e_1 - y^e_2)(x^e_3 - x^e_2)}, \\
N^e_4(x,y) &= \frac{(x^e_2 - x^e_1)(y - y^e_1) - (y^e_2 - y^e_1)(x - x^e_1)}{(x^e_2 - x^e_1)(y^e_4 - y^e_1) - (y^e_2 - y^e_1)(x^e_4 - x^e_1)} \\
&\quad \frac{(x^e_2 - x^e_3)(y - y^e_3) - (y^e_2 - y^e_3)(x - x^e_3)}{(x^e_2 - x^e_3)(y^e_4 - y^e_3) - (y^e_2 - y^e_3)(x^e_4 - x^e_3)}.
\end{aligned}
\tag{56}
$$

As displayed in Fig. 22, we construct the DNN representation of $N^e_1(x, y)$, the product of two linear functions with respect to x and y, by using the multiplication building blocks, i.e.,

$$
\begin{aligned}
&\mathcal{N}^e_1(x, y;\ \boldsymbol{x}^{e,*},\ \boldsymbol{\mathcal{A}}) \\
&= \mathcal{M}\Bigg(\mathcal{A}_0\Bigg(\frac{(x^e_3 - x^e_2)\mathcal{A}_0(y - y^e_2) - (y^e_3 - y^e_2)\mathcal{A}_0(x - x^e_2)}{(x^e_3 - x^e_2)(y^e_1 - y^e_2) - (y^e_3 - y^e_2)(x^e_1 - x^e_2)}\Bigg), \\
&\quad \mathcal{A}_0\Bigg(\frac{(x^e_3 - x^e_4)\mathcal{A}_0(y - y^e_4) - (y^e_3 - y^e_4)\mathcal{A}_0(x - x^e_4)}{(x^e_3 - x^e_4)(y^e_1 - y^e_4) - (y^e_3 - y^e_4)(x^e_1 - x^e_4)}\Bigg)\Bigg),
\end{aligned}
\tag{57}
$$

which is the function of nodal positions $\boldsymbol{x}^{e,*} = [\boldsymbol{x}^e_1, \boldsymbol{x}^e_2, \boldsymbol{x}^e_3, \boldsymbol{x}^e_4]$.

The other three element shape functions are constructed in the same manner. The summation of these shape functions, with nodal displacements as the weights of the last layer, becomes the interpolation in this element, as illustrated in Fig. 23. Once the shape functions in each element are represented by the DNN, the whole DNN interpolation of HiDeNN-FEM is obtained by assembling all DNNs together.

Following a similar process, we can construct the DNN representation for 3D element shape functions. The tri-linear element shape function associated with the first node of an 8-node element is

$$
\begin{aligned}
N^e_1(x,y,z) = &\frac{(x - x^e_7)(\tilde{y}^e_3\tilde{z}^e_8 - \tilde{z}^e_3\tilde{y}^e_8) + (y - y^e_7)(\tilde{z}^e_3\tilde{x}^e_8 - \tilde{x}^e_3\tilde{z}^e_8) + (z - z^e_7)(\tilde{x}^e_3\tilde{y}^e_8 - \tilde{y}^e_3\tilde{x}^e_8)}{\tilde{x}^e_1(\tilde{y}^e_3\tilde{z}^e_8 - \tilde{z}^e_3\tilde{y}^e_8) + \tilde{y}^e_1(\tilde{z}^e_3\tilde{x}^e_8 - \tilde{x}^e_3\tilde{z}^e_8) + \tilde{z}^e_1(\tilde{x}^e_3\tilde{y}^e_8 - \tilde{y}^e_3\tilde{x}^e_8)} \\
&\times \frac{(x - x^e_7)(\tilde{y}^e_8\tilde{z}^e_6 - \tilde{z}^e_8\tilde{y}^e_6) + (y - y^e_7)(\tilde{z}^e_8\tilde{x}^e_6 - \tilde{x}^e_8\tilde{z}^e_6) + (z - z^e_7)(\tilde{x}^e_8\tilde{y}^e_6 - \tilde{y}^e_8\tilde{x}^e_6)}{\tilde{x}^e_1(\tilde{y}^e_8\tilde{z}^e_6 - \tilde{z}^e_8\tilde{y}^e_6) + \tilde{y}^e_1(\tilde{z}^e_8\tilde{x}^e_6 - \tilde{x}^e_8\tilde{z}^e_6) + \tilde{z}^e_1(\tilde{x}^e_8\tilde{y}^e_6 - \tilde{y}^e_8\tilde{x}^e_6)} \\
&\times \frac{(x - x^e_7)(\tilde{y}^e_6\tilde{z}^e_3 - \tilde{z}^e_6\tilde{y}^e_3) + (y - y^e_7)(\tilde{z}^e_6\tilde{x}^e_3 - \tilde{x}^e_6\tilde{z}^e_3) + (z - z^e_7)(\tilde{x}^e_6\tilde{y}^e_3 - \tilde{y}^e_6\tilde{x}^e_3)}{\tilde{x}^e_1(\tilde{y}^e_6\tilde{z}^e_3 - \tilde{z}^e_6\tilde{y}^e_3) + \tilde{y}^e_1(\tilde{z}^e_6\tilde{x}^e_3 - \tilde{x}^e_6\tilde{z}^e_3) + \tilde{z}^e_1(\tilde{x}^e_6\tilde{y}^e_3 - \tilde{y}^e_6\tilde{x}^e_3)}
\end{aligned}
\tag{58}
$$

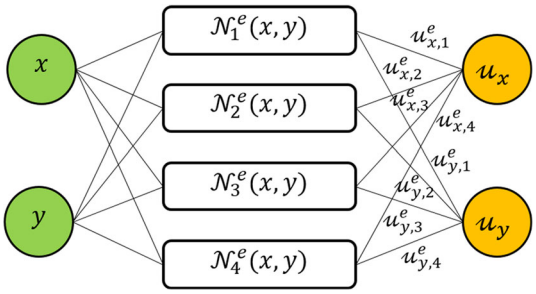


Fig. 23 Illustration for 2D HiDeNN inporlation in one element

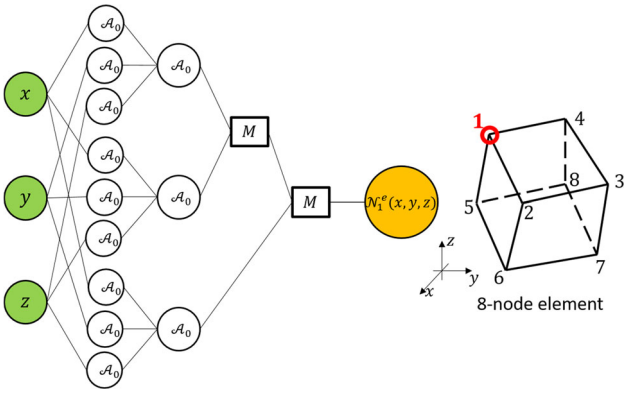


Fig. 24 Illustration for 3D element shape function $\mathcal{N}_1^e(x, y, z)$. The input is 3D physical coordinates (x, y, z)

with $\tilde{x}_k^e = x_k^e - x_7^e, \tilde{y}_k^e = y_k^e - y_7^e, \tilde{z}_k^e = z_k^e - z_7^e, k = 1, 3, 6, 8$. $\boldsymbol{x}_k^e = (x_k^e, y_k^e, z_k^e)$ as the coordinates of the k-th node of the element. The corresponding DNN representation of $N_1^e(x, y, z)$ is

$$
\begin{aligned}
&\mathcal{N}_1^e(x, y, z;\ \boldsymbol{x}^{e,*},\ \boldsymbol{\mathcal{A}}) \\
&= \mathcal{M}\Bigg(\mathcal{M}\Bigg(\mathcal{A}_0\Big(\frac{\mathcal{A}_0(x-x_7^e)(\tilde{y}_3^e\tilde{z}_8^e - \tilde{z}_3^e\tilde{y}_8^e) + \mathcal{A}_0(y-y_7^e)(\tilde{z}_3^e\tilde{x}_8^e - \tilde{x}_3^e\tilde{z}_8^e) + \mathcal{A}_0(z-z_7^e)(\tilde{x}_3^e\tilde{y}_8^e - \tilde{y}_3^e\tilde{x}_8^e)}{\tilde{x}_1^e(\tilde{y}_3^e\tilde{z}_8^e - \tilde{z}_3^e\tilde{y}_8^e) + \tilde{y}_1^e(\tilde{z}_3^e\tilde{x}_8^e - \tilde{x}_3^e\tilde{z}_8^e) + \tilde{z}_1^e(\tilde{x}_3^e\tilde{y}_8^e - \tilde{y}_3^e\tilde{x}_8^e)}\Big), \\
&\mathcal{A}_0\Big(\frac{\mathcal{A}_0(x-x_7^e)(\tilde{y}_8^e\tilde{z}_6^e - \tilde{z}_8^e\tilde{y}_6^e) + \mathcal{A}_0(y-y_7^e)(\tilde{z}_8^e\tilde{x}_6^e - \tilde{x}_8^e\tilde{z}_6^e) + \mathcal{A}_0(z-z_7^e)(\tilde{x}_8^e\tilde{y}_6^e - \tilde{y}_8^e\tilde{x}_6^e)}{\tilde{x}_1^e(\tilde{y}_8^e\tilde{z}_6^e - \tilde{z}_8^e\tilde{y}_6^e) + \tilde{y}_1^e(\tilde{z}_8^e\tilde{x}_6^e - \tilde{x}_8^e\tilde{z}_6^e) + \tilde{z}_1^e(\tilde{x}_8^e\tilde{y}_6^e - \tilde{y}_8^e\tilde{x}_6^e)}\Big)\Bigg), \\
&\mathcal{A}_0\Big(\frac{\mathcal{A}_0(x-x_7^e)(\tilde{y}_6^e\tilde{z}_3^e - \tilde{z}_6^e\tilde{y}_3^e) + \mathcal{A}_0(y-y_7^e)(\tilde{z}_6^e\tilde{x}_3^e - \tilde{x}_6^e\tilde{z}_3^e) + \mathcal{A}_0(z-z_7^e)(\tilde{x}_6^e\tilde{y}_3^e - \tilde{y}_6^e\tilde{x}_3^e)}{\tilde{x}_1^e(\tilde{y}_6^e\tilde{z}_3^e - \tilde{z}_6^e\tilde{y}_3^e) + \tilde{y}_1^e(\tilde{z}_6^e\tilde{x}_3^e - \tilde{x}_6^e\tilde{z}_3^e) + \tilde{z}_1^e(\tilde{x}_6^e\tilde{y}_3^e - \tilde{y}_6^e\tilde{x}_3^e)}\Big)\Bigg),
\end{aligned}
\tag{59}
$$

as illustrated in Fig. 24. Notice that the input is the 3D physical coordinates (x, y, z), and the DNN element shape function $\mathcal{N}_1^e(x, y, z;\ \boldsymbol{x}^{e,*},\ \boldsymbol{\mathcal{A}})$ is the function of nodal positions $\boldsymbol{x}^{e,*} = [\boldsymbol{x}_1^e, \boldsymbol{x}_2^e, \cdots, \boldsymbol{x}_8^e]$.

4 Numerical examples

In this section, several numerical examples are conducted to demonstrate the superior performance of the proposed HiDeNN-FEM framework. For illustrative purposes, 1D examples are used to investigate the convergence rate of HiDeNN-FEM (with inherent r-adaptivity), and examine the behavior of the rh-adaptivity of the HiDeNN-FEM by comparing it with the standard FEM. Specially, a 2D example is employed to further illustrate the capability of the HiDeNN-FEM for capturing the stress concentration by comparing it with the fine mesh from commercial software, such as Abaqus.

4.1 Problem statement

For illustrative purposes, a simple 1D example is presented here: consider a rod fixed at both ends under body force $b(x)$, i.e.

$$
\frac{d}{dx}(AE\frac{du}{dx}) + b(x) = 0, x \in [0, 10]; \tag{60}
$$

and Dirichlet boundary conditions

$$
u(0) = 0, u(10) = 0. \tag{61}
$$

Here, $u(x)$ is the displacement field, E is the stiffness of the rod, A is the section area and $b(x)$ is the body force.

Following the works of [17,18], a test function of v is multiplied on both sides of the Eq. (60) and integrated over the domain Ω,

$$
\int_\Omega v\left(\frac{d}{dx}\left(AE\frac{du}{dx}\right) + b(x)\right)dx = 0 \qquad \forall v\ with\ v = 0\ on\ \Gamma_u \tag{62}
$$

The associated weak form is given as

$$
\int_\Omega \frac{du}{dx}AE\frac{dv}{dx}dx - \int_\Omega b(x)v dx = 0 \qquad \forall v\ with\ v = 0\ on\ \Gamma_u \tag{63}
$$

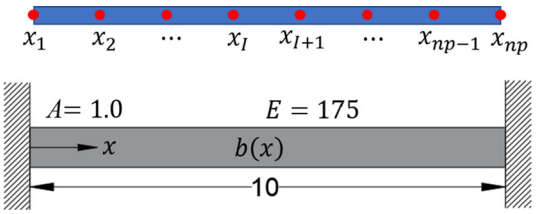


Fig. 25 1D numerical model for convergence study

After some organization, the equation, $J(u)$, was rewritten in minimum potential energy as

$$J(u) = \frac{1}{2}\int_\Omega AE(\frac{du}{dx})^2 dx - \int_\Omega ub(x)dx \qquad \forall u \in U, \tag{64}$$

where U is all the admissible displacement field. Finding minimum $J(u)$ gives the local displacement field. In HiDeNN-FEM, this process is formulated as an optimization problem as follows:

$$\begin{aligned} &\text{find} \quad u_I, \boldsymbol{x}_I^* \\ &\min \quad J(u) = \frac{1}{2}\int_\Omega \frac{du}{dx} AE \frac{du}{dx} dx - \int_\Omega ubdx \\ &\text{where} \quad u^h = \sum_{I=1}^{np} \mathcal{N}(\boldsymbol{W}, \boldsymbol{b}, x, \mathscr{A}) u_I = \sum_{I=1}^{np} \mathcal{N}(\boldsymbol{x}_I^*, x, \mathscr{A}) u_I \\ &\text{and} \quad \sum_{I=1}^{np} \mathcal{N}(\boldsymbol{x}_I^*, x, \mathscr{A}) = 1. \end{aligned} \tag{65}$$

The gradient descent method is applied to iteratively minimize $J(u)$ and solve for the local displacement field. Note that in the training procedure, the gradient of the objective function with respect to the variables (i.e. nodal coordinates and nodal displacements) is used to determine the update direction of the variables. The magnitude of the variables are computed by $u_{n+1} = u_n + \gamma_u \frac{\partial J}{\partial u}$ and $x_{n+1}^* = x_n^* + \gamma_{x^*} \frac{\partial J}{\partial x^*}$ iteratively, where n is the iteration number in the training, γ_u and γ_{x^*} denote the learning rate for training and are determined by an optimizer, such as Adams [31].

4.2 Convergence study of HiDeNN-FEM

The convergence of the proposed HiDeNN-FEM method is first studied and compared with the results obtained by standard FEM. The 1D example given in Fig. 25 is used as the objective for the investigation. The cross section of the bar is set to be $A = 1$, while the Young's modulus is defined as $E = 175$. For solving the optimization problem, np nodes are used to uniformly discretize the bar, and nodal coordinates serve as the initial positions for the r-adaptivity.

The body force is intensively designed as

$$b(x) = -\frac{4\pi^2(x-2.5)^2 - 2\pi}{e^{\pi(x-2.5)^2}} - \frac{8\pi^2(x-7.5)^2 - 4\pi}{e^{\pi(x-7.5)^2}}. \tag{66}$$

Substituting Eq. (66) into the governing equation of the 1D linear elasticity problem in Eq. (60) and coupling with the fixed boundary conditions, one can obtain the analytical solution for the displacement and its derivative as

$$\begin{aligned} u(x) = &\frac{1}{AE}(e^{-\pi(x-2.5)^2} - e^{-6.25\pi}) \\ &+ \frac{2}{AE}(e^{-\pi(x-7.5)^2} - e^{-56.25\pi}) - \frac{e^{-6.25\pi} - e^{-56.25\pi}}{10AE}x, \end{aligned} \tag{67}$$

$$\begin{aligned} \frac{du}{dx} = &\frac{2}{AE}(-\pi e^{-\pi(x-2.5)^2}(x-2.5)) \\ &+ \frac{4}{AE}(-\pi e^{-\pi(x-7.5)^2}(x-7.5)) - \frac{e^{-6.25\pi} - e^{-56.25\pi}}{10AE}. \end{aligned} \tag{68}$$

Figure 26 shows the analytical solution and the derivatives for the given body force. It can be seen that there are two peaks of different heights in the displacement and four peaks in the corresponding derivatives. The aim is to explore the performance of the proposed method for problems of multiple derivative peaks.

To study the convergence rate, the 1D bar is uniformly discretized by 23, 45, 89, 177, and 353 nodes, respectively. The program is developed in MATLAB R2019b, and the machine error is 2.2204×10^{-16}. The L_2 norm error and the H^1 error, defined as follows, are used to estimate the convergence rate:

$$||e||_{L^2} = ||u - u^h||_{L^2} = \frac{(\int_\Omega (u-u^h)^2 dx)^{\frac{1}{2}}}{(\int_\Omega (u^2) dx)^{\frac{1}{2}}}, \tag{69}$$

$$||e||_{H^1} = ||u - u^h||_{H^1} = \frac{(\int_\Omega (u-u^h)^2 dx + \int_\Omega (\frac{du}{dx} - \frac{du^h}{dx})^2 dx)^{\frac{1}{2}}}{(\int_\Omega (u^2) dx + \int_\Omega (\frac{du}{dx})^2 dx)^{\frac{1}{2}}}, \tag{70}$$

where u and u^h represent the exact displacement and the numerical displacement, respectively. The results of the L_2 norm error and the H^1 norm error are plotted in Fig. 27. To further quantify the convergence rate, the slopes and intercepts of the convergence curve are computed and compared with the results from the standard FEM. Tables 3 and 4 tabulate the slopes and intercepts of the four piecewise lines for L_2 norm error and H^1 norm error, respectively.

The following observations can be made based on the results:

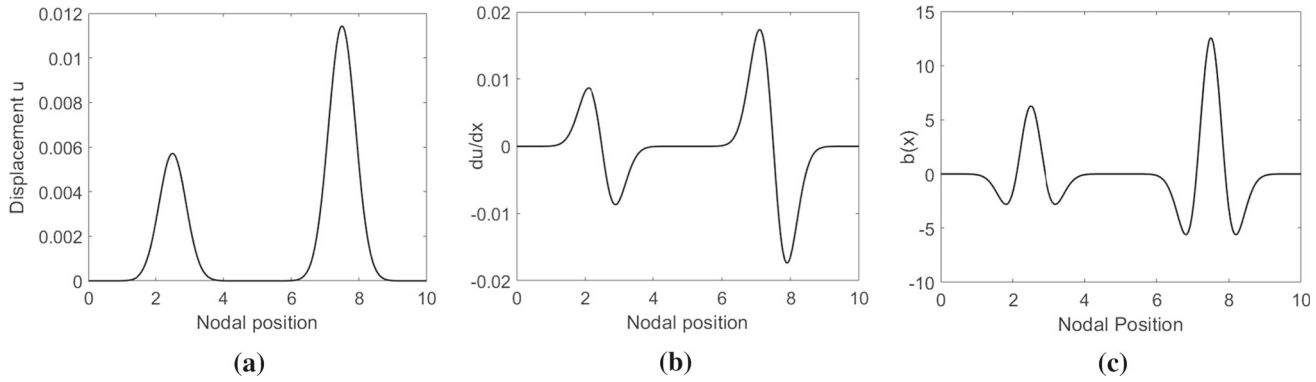


Fig. 26 1D numerical model for convergence study, **a** displacement **b** derivative of displacement, **c** body force

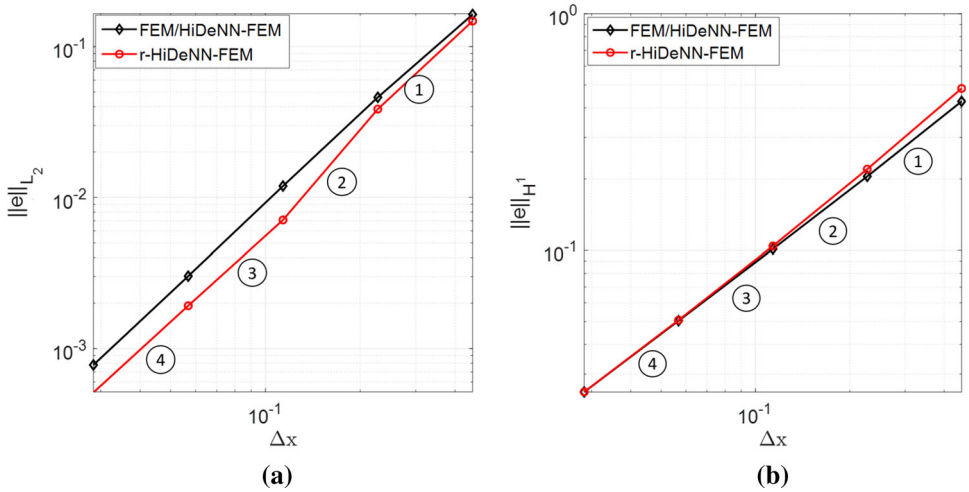


Fig. 27 Error estimation. **a** L_2 norm error comparison, **b** H^1 norm error comparison

Table 3 Convergence rate of L_2 error estimation, $log(||e||_{L_2}) \sim log(C) + klog(\Delta x)$

Refinement	FEM/HiDeNN-FEM		r-HiDeNN-FEM	
	Slope	Intercept	Slope	Intercept
1	1.83	−0.372	1.94	−0.382
2	1.95	−0.181	2.44	−0.362
3	1.97	−0.127	1.83	−1.692
4	1.96	−0.194	1.91	−0.774

Table 4 Convergence rate of H^1 error estimation, $log(||e||_{H^1}) \sim log(C) + klog(\Delta x)$

Refinement	FEM/HiDeNN-FEM		r-HiDeNN-FEM	
	Slope	Intercept	Slope	Intercept
1	1.054	−0.023	1.134	0.167
2	1.019	−0.074	1.080	0.087
3	1.005	−0.104	1.039	−0.002
4	1.001	−0.116	1.008	−0.092

- When the nodal positions are fixed, FEM and HiDeNN-FEM have the same convergence rate and accuracy.
- For a given number of nodes, the r-HiDeNN-FEM is more accurate than the standard FEM for the nodal displacements.
- The displacement accuracy is improved by r-HiDeNN-FEM as the nodes are uniformly refined.
- The normalized H^1 norm error of the r-HiDeNN-FEM converges to that by the standard FEM/HiDeNN-FEM when the nodes are uniformly refined.

Figure 28 illustrates the derivative of the displacement with 23 nodes. It can be observed that by learning the position of the nodes, the r-HiDeNN-FEM is able to move the nodes to the regions with large derivatives and capture the peaks of the derivatives. This implies that the HiDeNN-FEM can learn the loading conditions and intelligently adjust the nodes to achieve better performance.

4.3 rh-adaptivity by HiDeNN-FEM

In this case, the rh-adaptivity by HiDeNN-FEM is investigated. The 1D numerical example used in the previous case

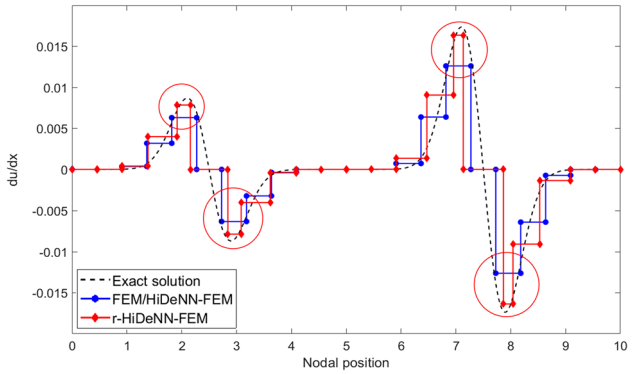


Fig. 28 Derivative of the displacement

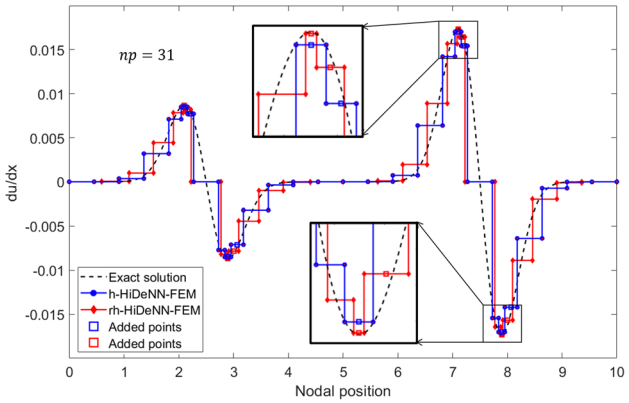


Fig. 29 Derivative of the displacement by rh-adaptivity through HiDeNN-FEM

is also used in the study of the rh-adaptivity, and the nodal number is set to 23. In HiDeNN-FEM, the rh-adaptivity is achieved by learning the objective $J(u)$, adding the neurons to the regions based on the back propagation, and moving the nodes to the regions through the derivatives of the loss function with respect to the nodal positions. This is because rh-adaptivity can be treated separately in the HiDeNN-FEM. The h-adaptivity in HiDeNN-FEM is achieved by adding neurons based on the nodal movement computed by the gradients of the loss function with respect to the nodal positions. For instance, if two neighbor nodes are moved close to each other, which implies that more degrees of freedom should be used between these two nodes, HiDeNN-FEM will add more neurons in this region and vice versa. Along with the movement of the nodes, another criterion is the gradient of the movement. We only add neurons to the region with fastest moving nodes. To avoid the inversion of the neighbor nodes, the minimum distance between the two neighbor nodes is constrained to 1/8 of the initial distance of the uniform distribution.

The updated optimization process for rh-adaptivity is given as follows,

$$\begin{aligned}
&\text{find} \quad u_I, \tilde{u}_K, \boldsymbol{x}_I^*, \tilde{\boldsymbol{x}}_K^* \\
&\min \quad J(u) = \frac{1}{2}\int_\Omega \frac{du}{dx} AE \frac{du}{dx} dx - \int_\Omega ubdx \\
&\text{where} \quad u^h = \sum_{I=1}^{np} \mathcal{N}(\boldsymbol{x}_I^*, x, \mathcal{A}) u_I + \sum_{K=1}^{nh} \mathcal{N}(\tilde{\boldsymbol{x}}_K^*, x, \mathcal{A}) \tilde{u}_K \\
&\text{and} \quad \sum_{I=1}^{np} \mathcal{N}(\boldsymbol{x}_I^*, x, \mathcal{A}) + \sum_{K=1}^{nh} \mathcal{N}(\tilde{\boldsymbol{x}}_K^*, x, \mathcal{A}) = 1 \qquad (71)
\end{aligned}$$

with nh new nodes $\tilde{\boldsymbol{x}}_K^*$ added in the DNN interpolation.

For rh-adaptivity, the nodes are moved based on the iterative learning until the loss function converges. For the purpose of comparison, the h-adaptivity only adds nodes to the domain based on the learning process, while the rh-adaptivity adds and moves nodes simultaneously.

Figure 29 presents the derivative of the displacement for the 1D problem by using the h-adaptivity and rh-adaptivity, respectively, through the proposed HiDeNN-FEM.

The following observations are made based on the results:

- h-adaptivity in HiDeNN-FEM is achieved by adding neurons based on the nodal movement computed by the gradients of the loss function with respect to the nodal positions. For instance, if two neighbor nodes are moved close to each other, which implies that more degree of freedoms should be used between these two nodes, HiDeNN-FEM will add more neurons in this region and vice versa. Along with the movement of the nodes, another criterion is the gradient of the movement. We only add neurons to the region with the fastest moving nodes.
- h-adaptivity in HiDeNN-FEM is able to capture the large gradients of the displacement and achieve better performance for the prediction of the displacement and its derivatives.
- Coupled with the r-HiDeNN-FEM procedure, the rh-HiDeNN-FEM is able to achieve even better results for the displacements and the sharp peaks of the derivatives.

Based on these observations, it can be concluded that the rh-adaptivity in the HiDeNN-FEM is accomplished by a learning process, in which increased resolution is obtained in the regions of large derivatives by adding nodes and moving nodes based on the training of the neurons. This leads to an intelligent discretization strategy without prior human knowledge.

4.4 2D example for capturing stress concentration

In this example, we will use the HiDeNN to solve a 2D problem with stress concentration by training the position of the nodes. Figure 23 presents a 2D bi-linear HiDeNN element

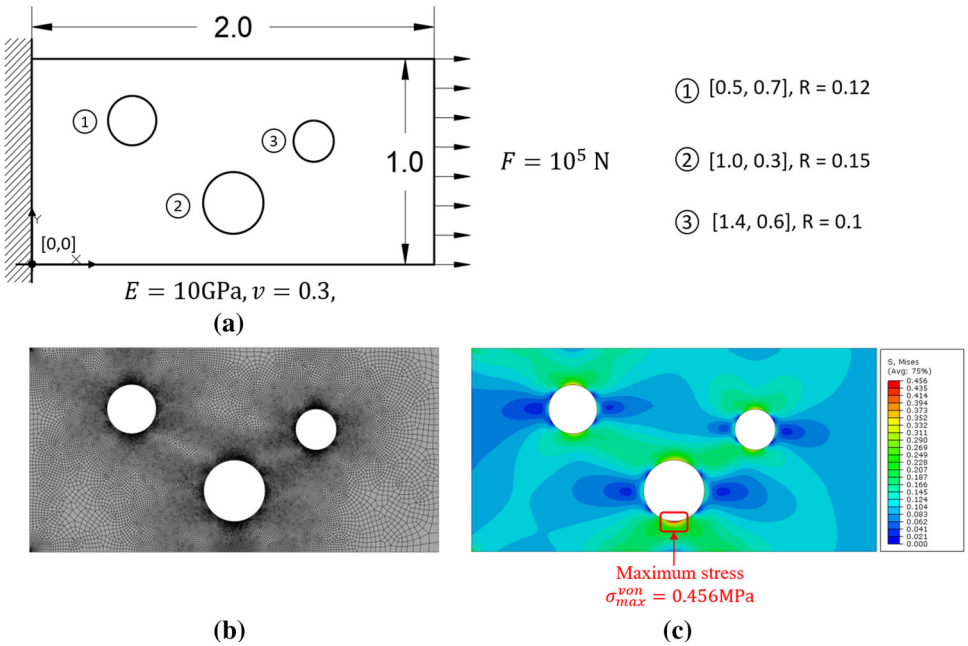


Fig. 30 Model and converged results of beam with three holes. **a** Model of the beam with three holes; values inside the parentheses denote the coordinates of the center and R is radius, **b** mesh of the converged results by Abaqus, **c** stress distribution after the convergence of the maximum stress from Abaqus; **d** 40×20 uniform mesh; **e** 60×30 uniform mesh; **f** 80×40 uniform mesh

constructed by using the proposed building blocks. As illustrated in the figure, the inputs of the HiDeNN element are the nodal coordinates, (x, y), while the outputs are the nodal displacements, $\boldsymbol{u}_x$ and $\boldsymbol{u}_y$. During the training, the nodal coordinates x_I^*, y_J^* are updated iteratively in order to obtain optimal solution accuracy.

In order to show how the HiDeNN trains nodal positions to intelligently capture the stress concentrations, an elastic tensile bar with three initial holes (refer to Fig. 30) under simple tensile loading is solved with plane stress condition. The dimension of the bar is **2 m** by 1 **m**. The left side of the bar is fixed while a uniform loading of **100 kN** is applied to the right side along the $+x$-direction. Young's modulus, E, of the bar's material is **10 GPa**, and Poisson's ratio, ν, is 0.3. The coordinates of the centers and the radii of the three holes are given in Fig. 30a. For comparison, a convergence study for the maximum local stress is conducted in Abaqus [32] and taken as the reference to examine the performance of the HiDeNN. The conforming mesh and corresponding stress solution from Abaqus are provided in Fig. 30b, c. The maximum local stress of the beam converges to **0.456 MPa** for **37928** elements. As illustrated in the figure, the maximum local stress occurs near the bottom surface of the largest hole. In order to capture the stress peak, extremely fine mesh is required at this region when using the standard FEM (refer to Fig. 30b).

For one-to-one comparison between the FEM and HiDeNN solutions, the domain of the bar is discretized with uniform quadrilateral elements. Three uniform meshes of 40×20, 60×30, and 80×40 are used (see Fig. 30d–f). Both HiDeNN and FEM are performed on these three meshes and compared against the conforming mesh solution.

The computed maximal stresses from FEM and HiDeNN, and their difference from the converged, conforming mesh solution are tabulated in Table 5. For FEM, although increasing the number of elements increases the maximum stress, the predicted value is still too low compared with the prediction from conforming mesh (33.99%, 27.85%, and 17.54% for three cases). On the other hand, the results obtained by HiDeNN show much better accuracy through learning the **optimal** nodal positions. As shown in Fig. 31, HiDeNN moves the nodes during training to the regions with stress concentrations. Even with the coarsest discretization, 40×20, and quadrilateral elements, HiDeNN is able to capture the maximum stress with less than a 2.5% difference from the converged value. For discretization of 60×30, 80×40, and 100×50, the differences are further reduced to 1.97%, 1.53%, and 0.88% respectively. The calculation time is affordable with an accuracy below 1%. The excellent performance of HiDeNN with pixel-like coarse discretization demonstrates the great potential of the HiDeNN of bypassing computationally expensive conformal mesh generation for image-based analysis. In concept, this is similar to isogeometric analysis (IGA), in that the difficulty of mesh generation is mitigated.

4.5 Outlook of HiDeNN

In this subsection, the general framework of HiDeNN is provided to show the flexibility and potential of this developed methodology for problems ranging from pure data analysis and semi-data investigation to the governing equation surrogate modeling. As mentioned in the introduction, the universal approximation achieved by deep learning has

Table 5 Summary of difference in maximum von Mises stress between the HiDeNN 2D solutions (uniform mesh) and conforming mesh solution from FEM

Type of analysis Conformal mesh	Number of elements 37928	Degrees of freedom	σ_{max}^{von}, MPa **0.456**	Difference -	Computational time (s)
Uniform mesh	800 (40 × 20)	1722	0.301	33.99%	-
HiDeNN 2D	800 (40 × 20)	3444	**0.446**	**2.19%**	**10.93**
Uniform mesh	1800 (60 × 30)	3782	0.329	27.85%	-
HiDeNN 2D	1800 (60 × 30)	7564	**0.447**	**1.97%**	**24.87**
Uniform mesh	3200 (80 × 40)	6642	0.376	17.54%	-
HiDeNN 2D	3200 (80 × 40)	13284	**0.449**	**1.53%**	**45.28**
Uniform mesh	5000 (100 × 50)	10302	0.419	8.11%	-
HiDeNN 2D	5000 (100 × 50)	20604	**0.453**	**0.88%**	**90.44**

Bold indicates the important values

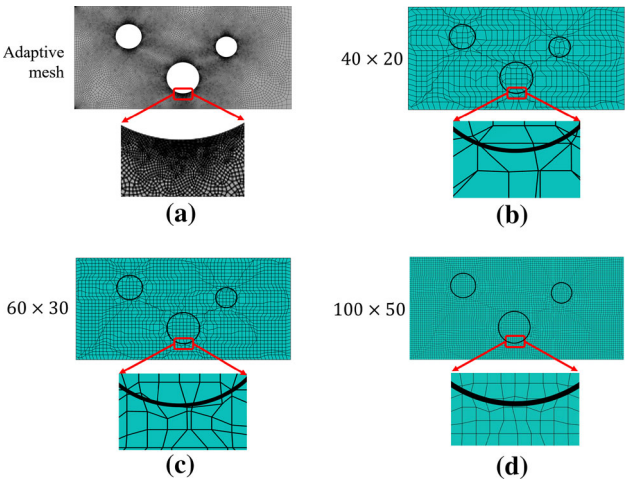


Fig. 31 Comparison between the conformal mesh and the HiDeNN discretization for stress concentration regions after learning the nodal positions. **a** Conformal mesh at the maximum stress regions; **b** HiDeNN discretization for 40 × 20; **c** HiDeNN discretization for 60 × 30; **d** HiDeNN discretization for 100 × 50

brought increasing attention to the ability of such methods to solve partial differential equations. The well-known methods in the literature include, but are not limited to, physics-informed neural network (PINN), the deep Ritz methods [33], deep Galerkin method(DGM) [16], etc. The general form of these methods can be described by Fig. 32. The input layer is the coordinates of the collocation points in the domain Ω and the time t, while the output is the solution of PDEs. The input layer and the solution layer are related by several layers of fully-connected neural networks. After the solution layer, several operators are used to calculate the loss function, where $\boldsymbol{I}$ is the identity matrix, and ∂ is the partial differential operator. The solutions are obtained by minimizing the residual of the governing equations and optimizing the weights and biases of the feed forward DNNs. The loss function is defined as the square summation of the residual of the governing equations and the boundary conditions.

The general HiDeNN framework is depicted in Fig. 33, including an input layer, a pre-processing layer, a customized DNN layer, a solution layer, and an operation layer. In HiDeNN, instead of being limited to the spatial and time space, the input is further extended to include a parameter space, expressed as $\Omega \times t \times D$, refer to Fig. 33. This parameter space D is used to formulate parameters of the physical systems, except for the scattered points and time. For instance, the parameters used in the formulation of the heat source model in additive manufacturing can be involved in the input vector for solving the temperature field of the printing process. The geometric constraint in optimization or the material properties related to spatial distribution can also be contained in the parametric space. After the input layer, a layer of pre-processing functions, $f_i(\boldsymbol{x}, t, \boldsymbol{p})$, is introduced to functionalize the input layer before sending it to the DNNs. The functionality of these functions is problem dependent. For instance, it can be used to transform the input variables into non-dimensional vectors for the training or perform a type of model order reduction for the input layer. During the training, these functions can be fixed or relaxed for training, depending on the specific problem.

Once the input is transformed or reformulated, it is taken as the input for the customized DNNs, and the solution is obtained by training the independent variables (i.e. nodal positions in this work when HiDeNN is degeneralized to HiDeNN-FEM). The DNNs are constructed in a hierarchical manner in which the response of each neuron can also be controlled by a sub-DNN. The depth of the hierarchy is determined by the investigated problem. In addition to being related to its upper network and communicating with other sub-DNNs, each sub-DNN can also connect with the output layer as part of the solution. The multiscale modeling of the DNN can be built upon this hierarchical manner and will be discussed in further work.

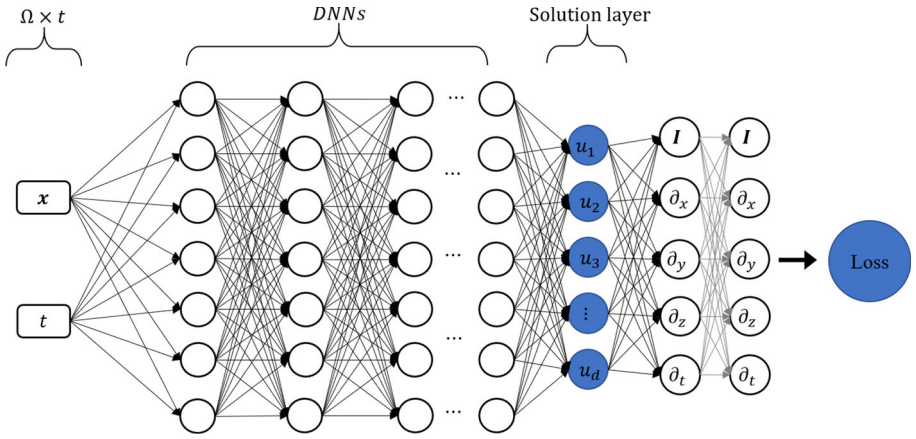


Fig. 32 Physical informed-deep learning methods for solving the partial differential equations

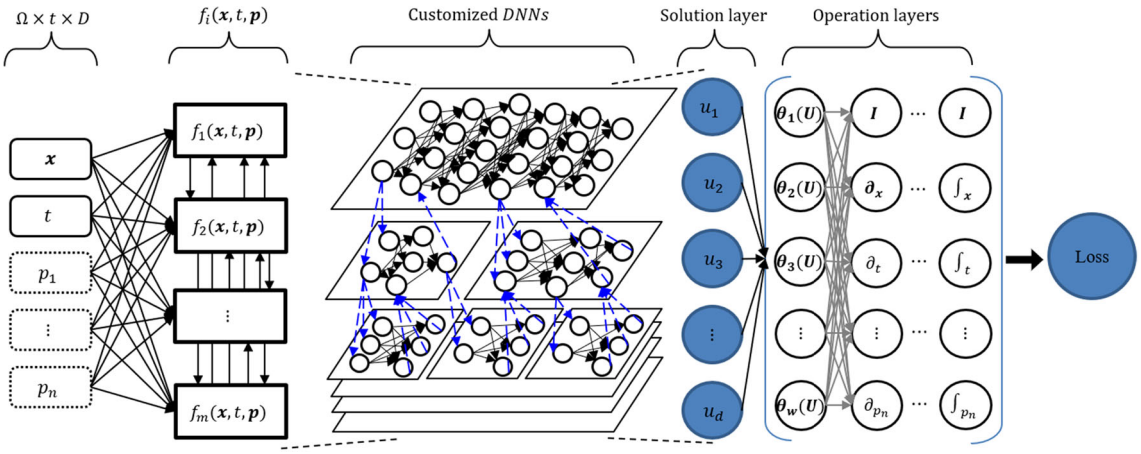


Fig. 33 A general framework of Hierarchical Deep-learning Neural Network (HiDeNN)

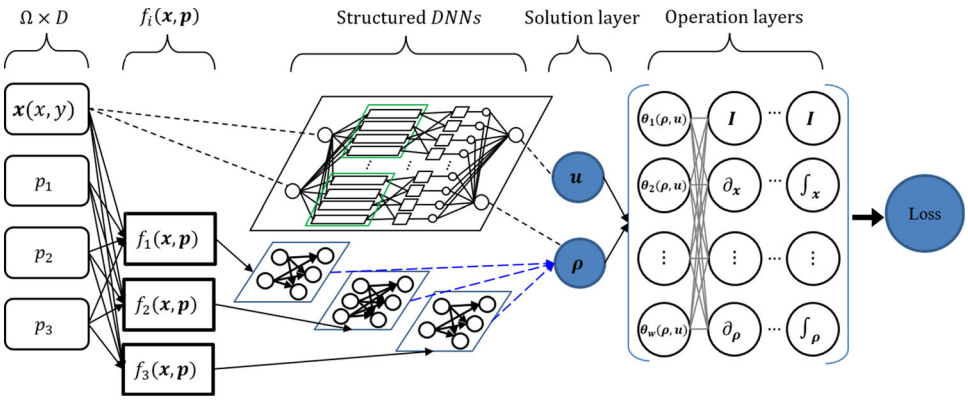


Fig. 34 Degenerated HiDeNN for topology optimization (HiDeNN-TO)

After the output of the solution layer, operation layers are introduced in order to accomplish scaling law or governing equation discovery in the training. For pure data analysis, the operation layers consist of representative functions (i.e., polynomials, splines, exponential functions, etc.), and the training is centered around finding the underlying law or patterns in the data sets. Functions $\theta_j (j = 1, ...w)$ are used to represent the potential functions for the discovery. When partial data and partial governing equation or scaling law is available, the operation layers are used to determine the coefficients of the governing equations and the potential relationships between the data sets and the proposed model. If the full model and the boundary conditions of the problem are available, the operation layers are used to construct the governing equations or the law, and the training process is performed as the surrogate modeling for real time prediction.

To illustrate the flexibility of the proposed HiDeNN framework, a HiDeNN based topology optimization is developed

to show a degeneralization of the HiDeNN. The degenerated HiDeNN for topology optimization (HiDeNN-TO in short) is illustrated in Fig. 34. Following the Solid Isotropic Material with Penalization method (SIMP) [34], p_1 represents the maximum allowable material density at a node, p_2 represents the minimum allowable material density at a node, and p_3 defines the total volume allowed in the final design. Using the pre-processing layer, three design constraints ($f_1(\boldsymbol{x}, \boldsymbol{p}) : \rho((x)) \leq \rho_{max}$, $f_2(\boldsymbol{x}, \boldsymbol{p}) : \rho_{min} \leq \rho((x))$, and $f_3(\boldsymbol{x}, \boldsymbol{p}) : \int_\Omega \rho(\boldsymbol{x}) d\Omega = V$) are integrated into the structured DNN (denoted by dashed arrows). The governing equation, density filter, as well as objective function for the problem of interests are formulated by the operation layers. For example, the functions $\theta_i(u, \rho)$ are used to formulate the filter, the differential operators are used for the governing equation and the integral operators e used for the objective function. The relationship among these operators can be easily formulated for problem of interests in topology optimization and will be proved in our next work.

5 Conclusions

In this work, a deep learning based numerical discretization scheme, the hierarchical deep learning neural network for finite element method (HiDeNN-FEM), is proposed to solve the partial differential equation. In the HiDeNN-FEM, the shape functions, widely used in standard FEM, are constructed by a structured deep neural network, and by training the DNN, HiDeNN-FEM can solve the partial differential equations. The weights and biases in the DNN are functions of input nodal coordinates, and thus, the training process in HiDeNN-FEM includes the optimization of the nodal positions. This is the spirit of the r-adaptivity, and it increases both the local and global accuracy of the interpolants. By fixing the number of hidden layers, the rh-adaptivity can be achieved by increasing the number of neurons through a predefined criterion. The additional nodes can thus be inserted by the training process, and the accuracy will be further improved. Numerical examples are performed using the HiDeNN-FEM for 1D and 2D linear elasticity problems. The convergence rate of the HiDeNN-FEM and standard FEM is studied, and the improved accuracy of the HiDeNN-FEM is apparent. This is due to the fact that nodes tend to move toward the peaks of the derivative of the solution for capturing the stress concentration at the largest derivative to the displacement. rh-adaptivity allows intelligent node insertion and movement, leading to better solution accuracy for the partial differential equations. Note that there is no manual control over the learning process, as the HiDeNN-FEM can understand the problem through training.

The generalization of the HiDeNN is obtained by introducing three elementary building blocks to construct hierarchical neural networks. The building blocks are linear functions, multiplication and inversion. By using these building blocks, the rational functions containing Lagrange polynomials, NURBS, IGA, and RKPM are constructed in a hierarchical manner by the simple interpolation functions. Furthermore, the enrichment function in the HiDeNN-FEM is achieved by the multiplication of neurons, which is equivalent to the generalized, extended, and partition of unity FEM. Leveraging the universal approximation capability of DNNs, the assembled hierarchical neural network can be trained to identify an optimal interpolation function to be used in the calculation process. This work is a concept proof and will be extended to learn multi-dimensional polynomials and arbitrary functions, space-time modeling, reduced order modeling, as well as topology optimization by constructing parameterized sub-DNNs in the future. In short, the HiDeNN will serve as a unification of various numerical methods that have broad application potential as illustrated in the outlook section.

Acknowledgements R. Domel is supported by Research Experience for Undergraduate supplement to U.S. National Science Foundation (NSF) grant number CMMI-1762035 (summer 2019). W. K. Liu and H.Y. Li are also supported by the same NSF grant. L. Cheng, J. Gao, and C. Yu conducted this research as an unfunded, exploratory (interdisciplinary) collaboration between Northwestern University and Peking University. L. Zhang, Y. Yang, and S. Tang are supported by National Natural Science Foundation of China grant number 11832001, 11988102 and 11890681. We would like to acknowledge the nice comments from Dr. Ye Lu.

References

1. Deng L, Dong Yu (2014) Deep learning: methods and applications. Found Trends Signal Process 7(3–4):197–387
2. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
3. Hornik K, Stinchcombe M, White H et al (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366
4. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2(4):303–314
5. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
6. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
7. Lee H, Kang IS, and (1990) Neural algorithm for solving differential equations. J Comput Phys 91(1):110–131
8. Meade AJ Jr, Fernandez AA (1994) The numerical solution of linear ordinary differential equations by feedforward neural networks. Math Comput Model 19(12):1–25
9. Lagaris IE, Likas A, Fotiadis DI (1998) Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 9(5):987–1000
10. Wu JL, Wang JX, Xiao H, Ling J (2017) A priori assessment of prediction confidence for data-driven turbulence modeling. Flow Turbul Combust 99:25–46
11. Xiao H, Wu JL, Wang JX, Sun R, Roy CJ (2016) Quantifying and reducing model-form uncertainties in reynolds-averaged navier-

stokes simulations: a data-driven, physics-informed bayesian approach. J Comput Phys 324:115–136
12. Weinan E, Han J, Jentzen A (2017) Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun Math Stat 5(4):349–380
13. Berg J, Nyström K (2018) A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing 317:28–41
14. Raissi M, Perdikaris P, Karniadakis GE (2017) Physics informed deep learning (part i): data-driven solutions of nonlinear partial differential equations. arXiv preprint arXiv:1711.10561
15. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
16. Sirignano J, Spiliopoulos K (2018) DGM: a deep learning algorithm for solving partial differential equations. J Comput Phys 375:1339–1364
17. Hughes TJR (2012) The finite element method: linear static and dynamic finite element analysis. Courier Corporation, Mineola
18. Belytschko T, Liu WK, Moran B, Elkhodary K (2013) Nonlinear finite elements for continua and structures. Wiley, New York
19. Höllig K (2003) Finite element methods with B-splines. SIAM, Philadelphia
20. Rogers DF (2000) An introduction to NURBS: with historical perspective. Elsevier, Amsterdam
21. Hughes TJR, Cottrell JA, Bazilevs Y (2005) Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput Methods Appl Mech Eng 194(39):4135–4195
22. Schillinger D, Dede L, Scott MA, Evans JA, Borden MJ, Rank E, Hughes TJR (2012) An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of NURBS, immersed boundary methods, and T-spline CAD surfaces. Comput Methods Appl Mech Eng 249:116–150
23. Cottrell JA, Hughes TJR, Bazilevs Y (2009) Isogeometric analysis: toward integration of CAD and FEA. Wiley, New York
24. Liu WK, Jun S, Zhang YF (1995) Reproducing kernel particle methods. Int J Numer Methods Fluids 20(8–9):1081–1106
25. Chen JS, Pan C, Cheng-Tang W, Liu WK (1996) Reproducing kernel particle methods for large deformation analysis of non-linear structures. Comput Methods Appl Mech Eng 139(1–4):195–227
26. Li S, Liu WK (2002) Meshfree and particle methods and their applications. Appl Mech Rev 55(1):1–34
27. Duarte CA, Babuška I, Oden JT (2000) Generalized finite element methods for three-dimensional structural mechanics problems. Comput Struct 77(2):215–232
28. Belytschko T, Moës N, Usui S, Parimi C (2001) Arbitrary discontinuities in finite elements. Int J Numer Methods Eng 50(4):993–1013
29. Melenk JM, Babuška I (1996) The partition of unity finite element method: basic theory and applications. In *Research Report/Seminar für Angewandte Mathematik*, volume 1996. Eidgenössische Technische Hochschule, Seminar für Angewandte Mathematik
30. Zhang L, Tang S, Liu WK (2020) Analytical expression of RKPM shape functions. Comput Mech 1–10
31. Chilimbi T, Suzue Y, Apacible J, Kalyanaraman K (2014) Project adam: building an efficient and scalable deep learning training system. In: 11th USENIX symposium on operating systems design and implementation (OSDI 14), 571–582
32. ABAQUS/Standard User's Manual (2016). Dassault Systèmes Simulia Corp, United States
33. Weinan E, Bing Yu (2018) The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. Commun Math Stat 6(1):1–12
34. Bendsoe MP, Sigmund O (2013) Topology optimization: theory, methods, and applications. Springer, New York